FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Natural User Interfaces in the Motor Development of Disabled Children

Pedro Miguel Lourenço Meleiro

DISSERTATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Pedro Amaral Rodrigues (PhD)

Co-Supervisor: João Tiago Pinheiro Neto Jacob

Co-Supervisor: Tiago Manuel Alves Pereira Marques

July 11, 2013

Natural User Interfaces in the Motor Development of Disabled Children

Pedro Miguel Lourenço Meleiro

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor António Augusto de Sousa
External Examiner: Doctor Pedro Miguel do Vale Moreira
Supervisor: Doctor Rui Pedro Amaral Rodrigues

July 11, 2013

Abstract

Natural User Interfaces have been widely used in the past few years in an ever growing range of contexts, including activities relying in body motion tracking, with an overall positive outcome. This thesis goals are to understand how to address a framework based upon body tracking devices and aimed at assisting children with motor impairments, as well as understanding what positive contribute it can deliver for their rehabilitation process.

A state of the art study regarding the most relevant devices and frameworks is addressed, and the most beneficial combination of these technologies is selected and detailed, including the emerged benefits and constraints. Follows the definition of a case study featuring two motor disorders that can take advantage of the technological specifications, as well as the types of exercise appropriate for this context.

The developed framework collects motricity data by asking the user to mimic the movements of a previously recorded exercise and is thoroughly detailed. The results obtained from the tests conducted during the validation process evidence that the data collected regarding the user performance is related with their impairment condition and denote certain motor patterns of the disorder, making it apt to be applied as an auxiliary tool for impairments diagnosis. However, a few detection and tracking issues in more complex exercises affect the relevance of the collected data, thus indicating that the technologies selected for this project can be applied in a real context to assist in rehabilitation sessions, but requiring additional evaluation metrics to support the obtained results.

Resumo

As interfaces naturais têm sido usadas ao longo dos últimos anos num cada vez mais vasto número de contextos, incluindo para atividades que recorrem captação de movimentos do corpo, com um resultado positivo. Os objetivos desta tese passam por perceber como uma ferramenta baseada em dispositivos de captação de movimentos poderiam contribuir no processo de reabilitação de crianças com incapacidades motoras.

É feito um estudo do estado da arte acerca da relevância dos dispositivos e ferramentas já disponíveis, bem como a combinação de tecnologias que poderiam resultar numa maior aproximação a esse objetivo e ainda as limitações que daí advêm. Segue a definição de um caso de estudo baseado em duas patologias motoras que, dadas as suas caraterísticas, estão aptas a tirar proveito das caraterísticas tecnológicas e a tipologia de exercícios mais propícia a este contexto.

A ferramenta desenvolvida procede à recolha de dados de motricidade com base na repetição de movimentos previamente gravados. Os resultados dos testes conduzidos na fase de validação são apresentados e evidenciam que os dados recolhidos vão de acordo com as limitações motoras reais de cada participante e denotam certos padrões das respetivas patologias, podendo ser aplicado como ferramenta auxiliar no diagnóstico. No entanto, algumas falhas na deteção dos movimentos em exercícios de maior complexidade tiveram implicações na relevância dos dados, indicando, assim, que o dispositivo escolhido pode ser aplicado no contexto da reabilitação motora, ainda que requerendo métricas de avaliação adicionais para suportar os valores obtidos.

Acknowledgements

I would like to start by thanking my supervisors Rui Rodrigues, João Jacob and Tiago Marques for their exceptional support throughout all these months. Their remarkable insight regarding Natural User Interfaces have been of the utmost importance in guiding me towards the right direction, as were their remarks and advices.

I would also like to thank my parents for their support and belief during all these years, hope someday I can pay them back. To my friends for putting up with me in all kinds of situations, including bad ones, thankfully there was always someone there to cheer me up regardless of what was bothering me.

Finally, I would like to thank the occupational therapy, physiotherapy and psychology specialists that provided support during key stages of the dissertation, especially Professor Catarina Grande, and Professor Joaquim Faias for establishing contact with the rehabilitation clinic, the outstanding availability and shared knowledge.

Pedro Meleiro

"Software is not limited by physics, like buildings are. It is limited by imagination, by design, by organization. In short, it is limited by properties of people, not by properties of the world."

Ralph Johnson

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Problem Definition . . . 3
- 1.3 Motivation and Goals . . . 3
- 1.4 Structure . . . 4

2 State of the Art **5**
- 2.1 Exergames . . . 5
  - 2.1.1 EyeToy: Kinetic . . . 6
  - 2.1.2 Wii Sports . . . 6
  - 2.1.3 Wii Fit . . . 7
  - 2.1.4 EA Sports Active . . . 7
- 2.2 Gesture Devices . . . 8
  - 2.2.1 2D video-based motion capture . . . 8
  - 2.2.2 Depth camera-based motion tracking . . . 9
  - 2.2.3 Motion capture with 3D position and orientation sensors . . . 10
  - 2.2.4 Comparing Gesture Devices . . . 12
- 2.3 Data Collection . . . 14
  - 2.3.1 OpenNI SDK . . . 14
  - 2.3.2 Move.me . . . 16
  - 2.3.3 Wii Remote libraries . . . 17
- 2.4 Conclusions and Related Work . . . 17

3 Methodology **19**
- 3.1 Technology Analysis . . . 20
- 3.2 Target Definition and Exercises Design . . . 20
- 3.3 Framework Development . . . 20

4 Technology Analysis **23**
- 4.1 Hardware Device Specifications . . . 23
  - 4.1.1 Joints tracking and accuracy . . . 24
  - 4.1.2 Field of view . . . 26
  - 4.1.3 Depth range . . . 27
  - 4.1.4 Environmental Conditions and Sensor Placement . . . 28
- 4.2 Software Specifications . . . 28
  - 4.2.1 Unity Game Engine . . . 29
  - 4.2.2 ZigFu Development Kit . . . 29

5 Target Definition and Exercises Design **33**
5.1 Targeting . . . 33
5.1.1 Spastic diplegia . . . 34
5.1.2 Hemiparesis . . . 35
5.2 Types of Exercise . . . 35
5.2.1 Schilder test . . . 35
5.2.2 Raising the arms above the head . . . 36
5.2.3 Sequence of poses . . . 37
5.2.4 Grabbing ball thrown by an external agent . . . 37
5.2.5 Bouncing a ball . . . 38
5.2.6 Side steps . . . 38
5.2.7 Scissors jump . . . 38
5.2.8 Conclusions . . . 39

6 Framework Architecture and Design **41**
6.1 Mechanics . . . 43
6.1.1 Execution Modes . . . 43
6.1.2 Evaluation on Keyframes . . . 43
6.1.3 Evaluation Metrics . . . 44
6.2 Play Modes . . . 45
6.2.1 Create Exercise . . . 45
6.2.2 Load Exercise . . . 47
6.2.3 Free Mode . . . 48
6.3 Data Storage and Retrieval . . . 48

7 Framework Implementation **51**
7.1 Exercise Creation . . . 51
7.2 Exercise Execution . . . 54
7.3 Data Storage and Retrieval . . . 57
7.3.1 User Data . . . 57
7.3.2 Exercise Data . . . 58
7.3.3 Exercise Keyframes Data . . . 59
7.3.4 User Execution Data . . . 60
7.3.5 User Collected Data . . . 60

8 Validation and Results **63**
8.1 Raising the arms above the head . . . 64
8.2 Side steps . . . 65
8.3 Sequence of poses . . . 66
8.4 Scissors jump . . . 67
8.5 Discussion . . . 67

9 Conclusions and Future Work **69**

References **71**

List of Figures

2.1 EyeToy: Kinetic gameplay screenshots [Gia] . . . 6
2.2 Wii Sports tennis and baseball screenshots [Ninc] . . . 7
2.3 Wii Fit statistics and gameplay screenshots [Nina] . . . 7
2.4 EA Sports Active gameplay screenshots [Qua] . . . 8
2.5 Vicon Motus 2D module using reflective markers [Con] . . . 9
2.6 Practical applications of structured light [TL] and time-of-flight [3dc] algorithms 10
2.7 MotionPlus hardware and rotation axes [Ard] . . . 11
2.8 OpenNI2 architecture overview [Opeb] . . . 15
2.9 Move.me diagram overview [McC11] . . . 16

4.1 Kinect components overview [Kina] . . . 24
4.2 Joints tracked by Kinect [Mic12] . . . 25
4.3 Skeletal tracking sample using OpenNI framework . . . 25
4.4 Arm overlapping the torso . . . 26
4.5 Field of view test with both arms down and raised . . . 27
4.6 Kinect's depth range in default and near modes [Kin12] . . . 27
4.7 Kinect's depth sensor behaviour to sunlight conditions . . . 28
4.8 Unity Editor [Unic] . . . 29
4.9 The required T pose detection . . . 30
4.10 Example of joint orientation issue regarding the head rotation . . . 31
4.11 Example of joint orientation issue regarding the right arm rotation . . . 31

5.1 Schilder test . . . 36
5.2 Raising arms test . . . 37
5.3 Poses test . . . 38
5.4 Side steps test . . . 39
5.5 Scissors test . . . 39

6.1 Block diagram overview . . . 41
6.2 Components diagram overview . . . 42
6.3 Exercise recording . . . 46
6.4 Exercise editing . . . 47
6.5 Paused execution . . . 47
6.6 Continuous execution . . . 48

7.1 Countdown time until exercise starts . . . 52
7.2 User database table . . . 57
7.3 Exercise database table . . . 59
7.4 User execution database table . . . 60

8.1 Test room floor plan 64
8.2 "Raising the arms above the head" exercise - average angles results on keyframes 65
8.3 "Side steps" exercise - average angles results on keyframes 65
8.4 "Poses sequence" exercise - average elapsed time and angles results on keyframes 66
8.5 "Scissors jump" exercise - average angles results on keyframes 67
8.6 Aggregated angle results 68

Abbreviations

API	Application Programming Interface
FOV	Field Of View
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HUD	Head-Up Display
IDE	Integrated Development Environment
IR	InfraRed
NUI	Natural User Interface
OS	Operating System
SDK	Software Development Kit
XML	eXtensible Markup Language
ZDK	ZigFu Development Kit

Chapter 1

Introduction

1.1 Context

Taking a look around us, it is easy to realize that nowadays people are steps closer to do different tasks with software and hardware systems in a more intuitive way than it was possible years ago, in a time when interacting with a computer would require the user to use standard devices such as the mouse and keyboard. Nevertheless, these peripherals still play a major role in interaction, as they have also been subject to many changes throughout the decades in order to provide, for instance, a more ergonomic experience, higher precision or wider control options. Despite some decades-old attempts at enabling other kinds of interaction with webcams or microphones [Pin11], both speech and gesture recognition were hardly reliable at the time, and so they never actually made it to the masses until the 90s, when faster-processing computers started being released at a frantic pace. This growth, along with existing research in HCI, created new standards in the way we interact, bringing usability and intuitiveness to more satisfying levels and thus attaining a larger audience. According to [HBC+96]:

> Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.

If we take, for instance, the pace touch-controlled smartphones and motion-controlled games have been growing lately [Cha12] [Lam12], it comes as no surprise that new kinds of interfaces are fully approaching the mass market and actually providing intuitive and comprehensive ways of interacting with software: touch sensitive surfaces are now commonly used for on-the-go devices such as smartphones, tablets and even handheld video game consoles, replacing the physical buttons that were once a standard in this kind of platforms, while gesture tracking is more popular indoors for both entertainment and serious games. Speech recognition is also playing a major role in interactivity and will not cease to grow in the coming years [Ber12], with Apple's Siri and Google's Voice Search being two of the most significant applications currently available. While being far from having wide appeal, it is even possible to navigate a computer by taking advantage

of the movement and blink of the eyes [Tob], though it is certainly a relevant approach for users with physical paralysis.

This kind of systems seeking to make human-machine interaction more natural and intuitive without requiring previous knowledge on how to use them, are called Natural user Interfaces. According to [Bla12]:

> A natural user interface is a user interface designed to reuse existing skills for interacting appropriately with content.

However, there is not a clear boundary separating natural from non-natural interfaces, i.e. an interface is considered more natural the less significant is the learning curve on how to properly interact with it. NUIs are usually designed to take advantage of valences inherent to the human body and adapting them to a technological context. We can easily infer, for instance, that drawing digitally using a touch-based tablet can be done in a more intuitive way than using a cursor-based mouse [Hof05], since the former makes for a more obvious approach to the elementary activity of hand drawing than the latter.

As was previously mentioned, natural user interfaces may suit many applications with considerable impact in the daily life. The potential of these interfaces in promoting physical exercise has been one of its most discussed and praised topics. By taking advantage of gesture devices, it is possible to deliver a software system that tracks a user's movements and compares them to pre-defined reference values in order to evaluate how he performed in a given task. The most commonly seen gesture-based software are intended for entertainment purposes, usually in a homely environment, but medical centers and retirement communities have been adopting this concept to stimulate people to spontaneously and regularly do exercise with great success [Sni08]. The appeal of low cost and widely available gesture devices has motivated researchers in understanding their impact as a tool for motor development [SS11]. Some authors, though, defend that the activity promoted by games with gesture devices does not compare with the real activity, energy expenditure-wise, thus not acting as a substitute to the outdoors activities but rather to sedentary activities [Dal09].

Nonetheless, this kind of interfaces have also been considered a key factor for the physical rehabilitation of motor impaired children, as it enables exercising in a more stimulating and autonomous way using videogames, as opposed to rehabilitation sessions, where patients are commonly asked to perform repetitive and tedious exercises [Rau08]. While rehabilitation software should not act as a replacement to assisted rehabilitation but rather a supplement, it brings benefits by allowing a child to exercise without requiring continuous assistance of a health care professional, who may not always be available. It also saves the patient the trouble of having to move from his house to the rehab center and vice-versa so often, which could be troublesome given the child's physical condition, by enabling the therapy to be partly done at home.

1.2 Problem Definition

There are, however, a number of limitations NUI-based exercising games commonly witness that prevent impaired children to fully take advantage of their intended goals. One of the causes is the focus on users without special needs, as a system that ignores the impairments of disabled children may inadvertently impose obstacles in terms of usability and accessibility, for instance, in the navigation menus, types of exercise or hardware device, thus contributing for an inadequate interaction design. Limited user input and data collection is another major drawback of current games: gathering no more than elementary data about the child, such as weight, height or age, will hardly be enough to propose an adequate set of exercises, with the software almost pointlessly selecting a set of appropriate exercises, or it may not even be programmed to do so. An inaccurate calculus of the progress pacing also brings undesired consequences; most exercising games do not adjust the difficulty of the challenges to a user's progress, which may lead to a frustrating outcome in case the exercises are too challenging, or a slower progress if the challenges do not require much effort. All the previously stated limitations account for what might result in an ineffective development of children with a certain motor disorder. Understanding how these limitations can be addressed is, therefore, of great relevance in order to prevent software aimed at disabled children from committing the same flaws.

1.3 Motivation and Goals

The chance to develop a software based on natural interfaces gives way to ground-breaking opportunities in establishing new standards on how people interact with a machine and broadens the number of tasks made possible with clear benefits over the preceding methods. Physical rehabilitation is only one of the many fields where software may intervene with positive results in providing a more cost-effective solution to traditional therapy sessions, while aiding disabled children with a more stimulating and effective recovery, conducting to improvements in the progress results over time, as suggested by [CCH11]. While natural interfaces can be a plus in terms of interaction, it is important to understand that peripherals tend to suffer changes over time in order to improve their reliability as a natural interface. It is then relevant to be aware of existing limitations and develop software accordingly.

This thesis aims to provide motor impaired children a system that enables physical exercising in a stimulating environment and with an adequate progress pace, while respecting their disabilities. Follows the delineated goals:

- Select a natural interface device that enables a child's movements to be tracked with satisfying results and apt for a wider range of impairments;

- Design and develop a framework based on the selected peripheral to collect motricity data allowing the system to compute what are the child's main physical disabilities;

- Deliver a case study by introducing an adequate set of exercises to validate the platform's effectiveness in the motor development of impaired children.

1.4 Structure

Following the introductory chapter, chapter 2 presents a study of the state of the art in natural user interfaces related topics, providing further information about the most relevant achievements of commercially available exergames. It is then succeeded by a section detailing multiple technological approaches to tracking user movements and the most significant devices. A comparison between devices based upon multiple parameters is also conducted and thereafter the frameworks and libraries for interpreting data obtained from the gesture devices are detailed. Conclusions and related work regarding this stage are detailed later in this chapter.

Chapter 3 addresses the methodology and approach taken during the development process, referring to the relevance of each of the stages and how they are related with the other stages.

The following chapters describe in greater detail the development process stages. Chapter 4 delivers a technological analysis, both in terms of hardware and software specifications and limitations, along with some tests performed in order to further explore possible constraints.

Chapter 5 proposes a target-audience for the case study proposed in this thesis, along with functional tests regarding the types of exercise that would better fit the technological context. This stage counted on the collaboration of an Occupational Therapy specialist that provided great insight regarding the motor rehabilitation field.

The next two chapters, chapter 6 and 7, make up for the last stage of the development process. The former details both the framework architecture and design on a higher level, and the latter goes into the implementation details, challenges that surfaced and how they were conducted.

The validation process and obtained results are addressed in chapter 8, which details the test group and obtained results. The contribution and relevance of the tests outcome in the assisting children in their motor rehabilitation is also discussed.

Finally, chapter 9 sums up the work, drawing conclusions about the significance of the obtained results, as well as possible ways of improving and extending the framework to other case studies and suggested research on a similar technological context.

Chapter 2

State of the Art

This chapter delivers a study regarding the state of the art in Natural User Interfaces applied to the context of physical activity and user data collection. Regarding the former, this chapter provides an overview of currently available exercising games, the techniques used to retrieve user data and evaluation metrics; also addressed is the hardware devices that enable to track the user movements, as well as a comparison of these technological solutions based on a set of parameters. About the user data collection topic, the second section focus upon the frameworks responsible for using the information collected by the hardware devices. Conclusions drawn from this study are addressed in the last section of this chapter.

2.1 Exergames

Exercising games have been around for more than three decades now. Early attempts aimed at turning videogames into a more active and healthy hobby by bringing new ways of play incorporating more body motion than that of the fingers to press buttons. The Joyboard for Atari 2600 and the Power Pad for the NES system were two of the hardware peripherals that actually reached the market during the 80s [Bog07] [Koh06]. However, these attempts at placing physical buttons inside a differently shaped controller and aimed at establishing new trends in the video gaming industry were inaudible and quickly forgotten until 1998 with Konami's Dance Dance Revolution. It uses a 4-button dance pad and includes a playlist featuring dozens of themes; players interact with the game by rhythmically taping the pad's buttons with their feet, following a pre-defined sequence [DDR]. This Japanese series was met with a notable worldwide success, being one of the most influential products in the exergaming genre during the 90s and early 00s.

Modern time exergames, though, provide increased complexity and more user-centered approaches at exercising. This section will focus on four of the most relevant software products, underlining what and how input data is collected, the focus on user progress, gesture devices utilized and effectiveness in promoting exercise.

2.1.1 EyeToy: Kinetic

One of the first great successes in the exergaming genre was released for the Sony PlayStation 2 system using the EyeToy camera, which is essentially a webcam to be used with the platform. Kinetic captures the movements of a player in order to evaluate his performance regarding four different groups: mind and body, cardio, combat and toning [Plaa]. Also featured is the Personal Trainer mode, a 12 week programme that enables user progress to be saved over time, with sessions taking place on a daily basis (though it is not mandatory to do all sessions) [Rop05]. A virtual trainer recommends the exercises the user should take and, by the end of each routine and each week, the performance is evaluated by a grading system ranging from A to E. When creating a profile, the game asks some information about the player: age, weight, height and how often the user exercises. However, the software does not authenticate this input data, despite adapting the workout intensity of the exercises [Rop05]. Nevertheless, being based on a 2D-video motion capture system, similar to a standard webcam, the challenges are designed in such a way that an accurate body tracking is not what it pursues, but rather the two-dimensional silhouette produced by the body shape, thus aiming for a more shallow yet entertaining approach to the exergaming genre.

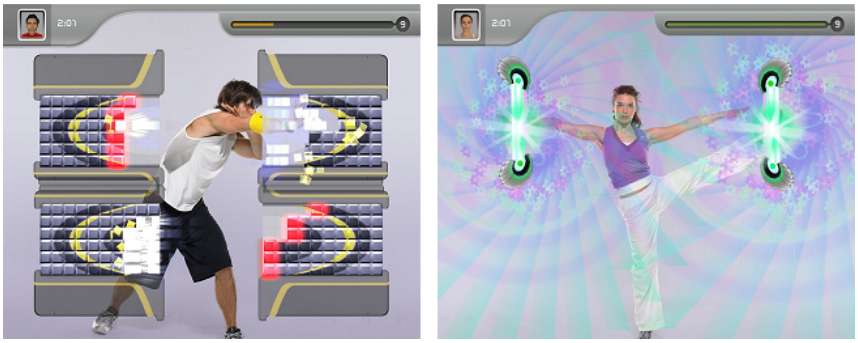

Figure 2.1: EyeToy: Kinetic gameplay screenshots [Gia]

2.1.2 Wii Sports

Commonly stated as one of the most influential games [Koh09] [Sli12], Wii Sports was released simultaneously with the Wii Remote back in 2006 and features five different sports: tennis, baseball, golf, bowling and boxing [Nin06]. It collects only minimal user information - left or right handed - but adjusts the difficulty of the challenges (i.e., the artificial intelligence of computer-controlled opponents) in a way similar to a handicap system, depending on how a player performs, thus balancing the exercises to a suitable level [Nin06]. It also delivers statistical data regarding the player's progress over time and a daily "fitness age" test, which assigns an age depending on the performance [Ninb]. Despite using three-dimensional position and orientation for a more thorough evaluation, it only tracks the hand holding the Wii Remote device, or the two hands using the Nunchuk add-on in the boxing activity, thus redesigning and simplifying gameplay in a way

that the featured sports require no more than upper limbs movement. For example, mimicking the racket movement in tennis, or swinging the bat and throwing the ball in the baseball activity.

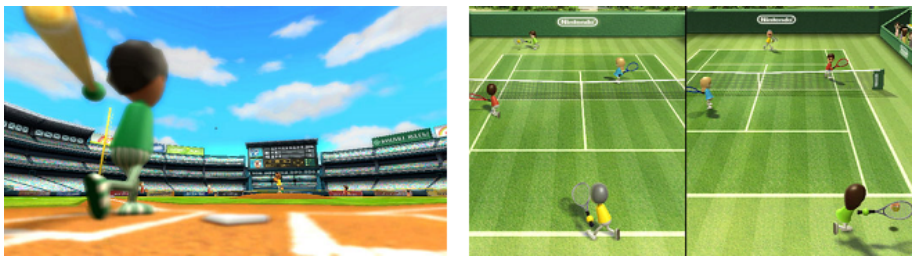

Figure 2.2: Wii Sports tennis and baseball screenshots [Ninc]

2.1.3 Wii Fit

In terms of data collected from the user, Wii Fit is a step forward when compared to previously addressed exergames. It is divided into four distinct categories - yoga, strength, aerobics and balance -, each of these featuring multiple exercises [Nin08a]. When starting a new profile, it asks the user to input data such as age and height; however, the software is actually capable of measuring the weight and center of balance using the Wii Balance Board device, an electronic balance board platform featuring four pressure sensors and withstanding up to 300 Kg [Nin08b]. The board device is wirelessly connected to a Wii system and the player must stand upon the platform during game sessions. Periodical body tests evaluate the player's body mass index and balance control. Similarly to Wii Sports, Wii Fit also features a fitness age test based on progress [Nin08a]. Given the more slow-paced nature of its exercises, Wii Fit has been successfully used for physiotherapy rehabilitation of both young and elderly people, most notably for body balance control, since the Wii Balance Board provides unique capabilities in this field [dSMPL$^+$12].

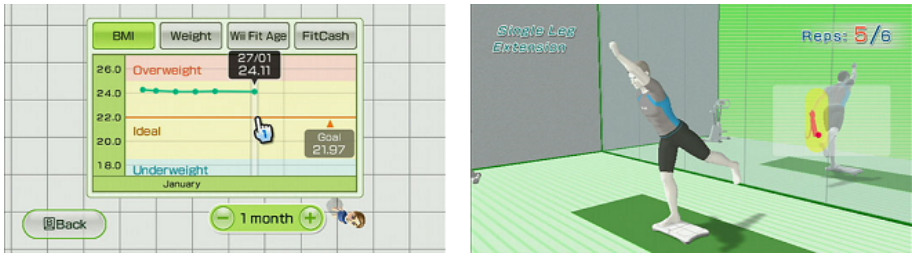

Figure 2.3: Wii Fit statistics and gameplay screenshots [Nina]

2.1.4 EA Sports Active

With the growing popularity of exergaming, mostly due to the notable success of the Wii line series, Electronic Arts releases its own version based on Wii Remote with the Nunchuk add-on, and optionally using the Wii Balance Board for an enhanced experience [Ele]. EA Sports believed

that Wii Fit's exercises typology were a better fit for the eastern market due to its stretching and balance based nature, so the company intentionally adopted a more weight loss and intensive workout approach in order to bring a product fully aimed at a western audience [Tho08].

The main focus of this software is the 30 Day Challenge mode, which proposes to the user a set of customizable exercises for that day; it then reports the projected performance and how well the daily goals were delivered [Tho08] [Ele]. It also features a Journal option that enables the user to answer nutrition and activity surveys, which then will add them to a virtual calendar and stimulate the player to achieve daily landmarks. It is also possible to set goals in terms of calories burned, hours spent doing exercise or number of workouts [Tod09]. Besides, it tracks progress over time specifically for each body zone and stimulates the players to achieve landmarks by imposing exercising goals [Ele].

Figure 2.4: EA Sports Active gameplay screenshots [Qua]

2.2 Gesture Devices

For exergames to work correctly, though, at least one tool is needed with the ability to track a person's movements and determine what gestures are being performed. These are called gesture devices and can adopt various approaches in order to do the recognition. The previously seen Joyboard and Power Pad peripherals are two of the earlier input devices attempting to do so by determining the feet movements with force feedback in the former and fixed points position in the latter, though their purposes could be easily bypassed and intentionally misused. This section will focus on some of the most prominent technologies for motion capture accordingly to the relevance of their current practical applications and technological achievements, more specifically, motion capture based on 2D video camera, depth camera and position and orientation sensors.

2.2.1 2D video-based motion capture

Capturing motion using 2D video capture technology is one of the earlier case studies in terms of motion capturing techniques, since it is usually based on fairly proliferated hardware devices, such as recording cameras and similars. The PS2's EyeToy camera used by EyeToy: Kinetic is one such device; though it is based on a single camera, multiple cameras can be used to improve

tracking results, though they tend to get more expensive and roomy in order to obtain an interesting outcome [SB06].

2.2.1.1 Webcam standalone

A webcam is a video camera that connects to a computer generally through an USB port. Being available since the 90s, it is commonly used for video calling, video recording, digital photography, security surveillance and computer vision. Regarding the latter, and given the popularity of webcams, there a great number of libraries and frameworks intended for this kind of devices which includes motion tracking algorithms, thus enabling a webcam to be used for augmented reality applications or gaming. Specific light conditions are required for a webcam to operate correctly.

2.2.1.2 Webcam with markers

Another approach for body tracking with 2D video cameras is to visibly place markers throughout the user's body parts in order to enable gesture recognition libraries to track the body motion more accurately. The markers may range from a lot of different objects; some developers suggest the use of LEDs, QR (or other kind of simplified) codes, etc. Assigning a specific color/code for each body part may ease the complexity of the algorithms and thus making for a more precise detection. A comparison between different markers-based tracking methods is conducted in [Ric99], featuring systems relying on reflective markers for improved results, such as the Vicon Motus.

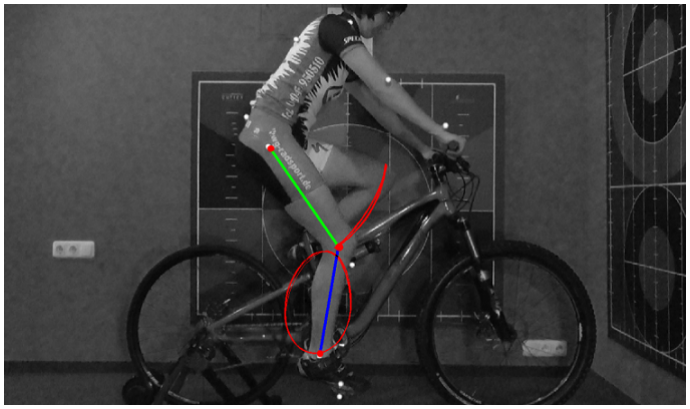

Figure 2.5: Vicon Motus 2D module using reflective markers [Con]

2.2.2 Depth camera-based motion tracking

Depth cameras create 3D representations of what it records using techniques and algorithms that generate a depth map. This could lead one to believe that at least two cameras are needed in order to obtain stereoscopic images and, although that combination can be used to obtain improved results, that is not a requirement since time-of-flight or structured light cameras are capable of

producing depth images and without requiring traditional computer vision algorithms [Han12]. The Kinect sensor is one widely known device enabling motion tracking using structured light process.

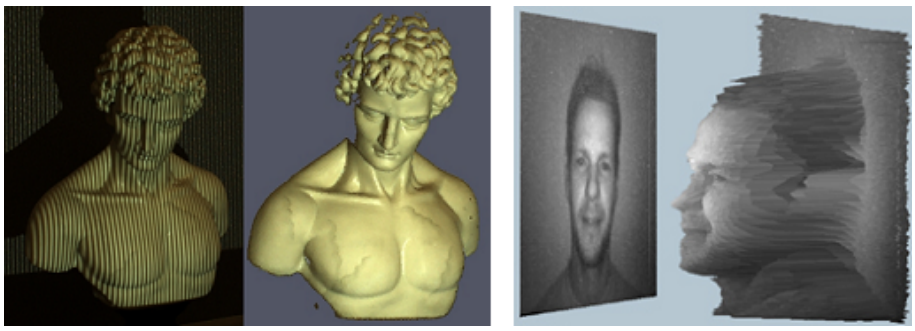

Figure 2.6: Practical applications of structured light [TL] and time-of-flight [3dc] algorithms

2.2.2.1 Kinect

Kinect is a device by Microsoft initially released for Xbox 360 video game console and later for Windows. Besides the RGB camera, it features a depth sensor, consisting of an infrared laser and an image sensor, which captures 3D video data to calculate the depth in an image [Bar10] [Opea]. After computing a depth map using structured light with depth from focus (i.e. an object distance depends on its blurriness) and depth from stereo's parallax (i.e. an object closer to the camera shifts more to the side) techniques, it infers the body position using machine learning, first transforming depth image into body part image and then the body part image into a skeleton [SFC+11].

2.2.3 Motion capture with 3D position and orientation sensors

Another relevant, inexpensive and accurate approach on tracking human movements is using controller-based sensors allowing for 3D position and orientation. Due to the success of the Wii and subsequent motion-enabled video gaming consoles, these tools have reached mainstream status and proved to be effective in tracking both position and orientation by combining accelerometers and gyroscope technologies and thus providing a recognition with six degrees of freedom, i.e. free backward/forward, up/down and left/right movement along with three-axis rotation.

2.2.3.1 Wii Remote and Wii Remote Plus

The standard Wii Remote device was originally released in 2006 with the Nintendo Wii video game console. It contains an accelerometer for motion input that has the ability to sense acceleration along three axes and an optical sensor that enables the Wii system to determine where the Wii Remote is pointing to. The device's image sensor enables it to sense the light of infrared LEDs of a sensor bar, placed near the display device; this bar features two clusters of LEDs - each of

them placed near each of the bar's edges. Since the distance between these two sets of lights is a fixed, and given that the image sensor processes the distance based on the light emitted by both clusters of LEDs of the sensor bar, it makes it possible for the Wii's CPU to compute the distance between the Wii Remote's current position and the sensor bar, thus enabling 3D positioning using triangulation.

However, accelerometers cannot distinguish between linear movement and gravity, which means it can only measure linear acceleration of the device effectively if there's no rotation. In order to measure rotation it has to relinquish the ability of tracking horizontal movement, and even so it cannot measure horizontal rotation [Car08]. For this reason, the technology included with the standard Wii Remote device cannot be used for real time 3D orientation and positioning in a given scenario. A way to work around this restriction is to include gyroscopic technology; this was done in 2009 with the MotionPlus add-on to the Wii Remote and in 2010 with a substitute to the combination of both, the Wii Remote Plus [Inv08]. Gyroscopes measure all kinds of rotation in a clean and responsive way, but lack the ability to calculate linear movement; with the combination of both technologies, the improved Wii Remote is able to apply gyroscopes for rotation and accelerometers to calculate linear movement using the former's tracked information [Car08], thus enabling six degrees of freedom with notable results - commonly referred to as 1:1 motion detection [Cha].

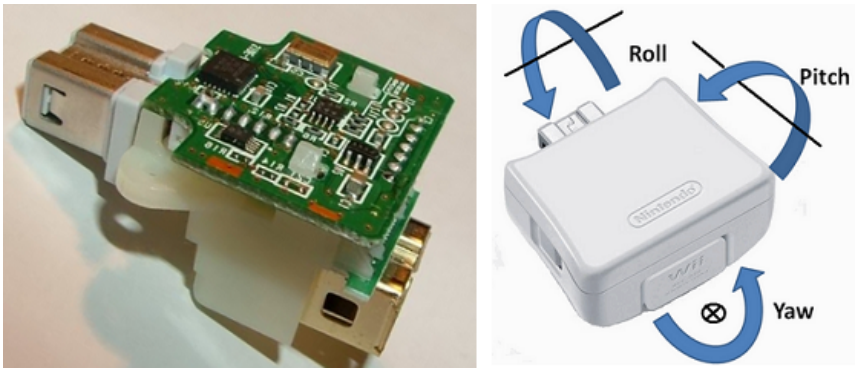

Figure 2.7: MotionPlus hardware and rotation axes [Ard]

2.2.3.2 PlayStation Move with PlayStation Eye

In terms of 3D positioning, PS Move follows a different direction, as the solution developed by Sony is almost the "reverse" of the one seen in Wii Remote Plus. Instead of having a sensor emitting lights placed near the display device, the PS Move's device does so by having a light attached to its top. The PS Eye, a webcam-like camera for the PS3 system, by assuming the position of the Wii Remote's sensor bar, tracks the position of the device in the flat 2D-image and determines its distance from the camera by measuring the size of the light in the captured image [Kum09].

Similarly to the Wii Remote Plus, PS Move relies on a three-axis accelerometer and gyroscope to measure rotation and linear movement, as well as a magnetometer for calibration of cumulative errors [Mik09]. A second controller for two hands tracking can be used along with the first one, whether a PS Move or a Navigation Controller (the equivalent to a Wii Remote's Nunchuk controller).

2.2.4 Comparing Gesture Devices

Most of the discussed devices rely on several approaches that fully or partially enable human body recognition and may fit better or worse depending on a system's context and its goals. Thus, it is relevant to measure the impact of each technology in a successful deployment of a system aimed at children with motor disabilities, for instance: will they be able to experience it without having to constantly move places (e.g. home-clinic-home), will the device allow an effective recognition, will it enable a child to practice the impaired body zone over time or will it be usable and accessible. In order to do so, it is relevant to conduct a comparison based on a set of parameters that exploits the benefits and drawbacks carried by each of the gesture devices previously described.

2.2.4.1 Price and availability

This parameter intends to compare devices based on average price in regular marketplaces and their availability to the target market, given its relevance in understanding if the aimed device is affordable and can be easily found. Regarding this matter, webcams are both the less expensive, with a mid-range quality camera costing between 30 Euros and 40 Euros, and the more widely available, as it serves several applications, has low manufacturing cost, high flexibility and are commonly build into personal computer screens. The markers to be used with a webcam may have a varying cost, depending on the material, but its cost is mostly residual; however, the markers would need to be made available with the software, since there is no standardized codes in the market to serve the specific purpose of body recognition (e.g. it may vary according to the number of markers we need, the developed algorithms, etc). Both the Wii Remote Plus and PS Move devices have similar price tags, ranging from 40 Euros to 70 Euros, depending whether the investment is for the standalone devices or also the accessories (sensor bar and Nunchuk in the former, PS Eye and Navigation Controller for the latter) and are commonly available in the videogames and informatics stores. Nintendo stopped selling Wii Remote devices little after the Wii Remote Plus made its debut, making it harder to find them in common marketplaces. With a suggested retail price of 250 Euros and availability restricted to selected retailers, Kinect for Windows is more expensive than any other device.

2.2.4.2 Existing libraries and support

In order to develop the platform, it is relevant to use a pre-existing library or SDK that provides a set of methods to communicate with the hardware and software components, as well as to streamline some tasks, enabling to focus development on what really matters. Kinect has been receiving

considerable attention by developers, given its effectiveness in tracking the human body, and continuous development by Microsoft with the official SDK is being conducted aimed at speeding up development, extending depth data or improving support. Other frameworks include OpenNI, which supports depth camera devices other than Kinect and provides a set of open source APIs and an active community forum with support by developers. By having a Bluetooth connection, the Wii Remote has seen quite a few compatible libraries and frameworks over time, with one of the most relevant being WiimoteLib; however, most of them have not received any updates to fully support Wii Remote Plus and, right now, the only way to get it fully working is with an official development kit, which is only made available to selected game developers. PS Move is restricted to Sony's official application server, Move.me, which has an additional purchase cost for non-academic users, thus narrowing the amount of active developers and community support.

2.2.4.3 Hardware requirements and dependencies

Another significant aspect to consider when selecting a device is to acknowledge what are the system requirements and dependencies, mainly for the end consumer. A webcam usually requires low hardware system requirements, though it may be optimized if running on a machine with greater processing power and more memory RAM; a standard webcam connects to a computer using an USB port. The Wii Remote and Wii Remote Plus device also require minimum system specifications and plugs to a computer via Bluetooth. Kinect for Windows, though, can only be used with Windows 7 or Windows 8 operating system, according to Microsoft's imposed license terms, and requires a computer with at least 2GB RAM and a dual-core 2.66 GHz or faster processor [Kinb]. The PS Move device can only be used for PS3 software and needs approval by Sony before making it available, thus a PS3 system is required both for development and end users [Plab].

2.2.4.4 Target audience coverage

Equally important is to select a device that can reach the broadest number of children motor impairments, while keeping the effectiveness of covered ones. This parameter is a clear drawback for devices that enable only partial body recognition, such as Wii Remote, Wii Remote Plus and PS Move, even if more than one can be used to get data from both upper limbs. Either webcam or Kinect, regardless of how they perform, can track full body motion.

2.2.4.5 Ease of use

Regarding usability matters, it is relevant to consider if and how well a device addresses ease of use to all covered children impairments, preventing them from performing naturally due to hardware. Research reveals that using devices that require direct contact with the user end up being an obstacle to perform more effectively, given that in a different context most motor exercises would not usually require that kind of tools. Both webcam standalone and Kinect do a body-free tracking experience, unlike both Wii Remote devices and PS Move, which require the child to

hold at least one controller, and webcam using markers attached along the body, though it may be done in an almost seamless way.

2.2.4.6 Tracking effectiveness

Finally, one major step is comparing how effectively the devices can be used for this purpose, more specifically, what are the constraints and limitations of the hardware being used. Although this may vastly rely on how the algorithms used by software take advantage of the hardware, truth is a device's specifications play a major role in the time and mathematical complexity of, mainly, computer vision algorithms. Existing projects and research demonstrate that using a webcam for body tracking is by far the least reliable solution, given hardware limitations; even using markers, motion recognition is subject to common errors and inconsistencies. Kinect, in the other hand, has seen favourable research results in precise body tracking and has been previously applied to exercising contexts, even if latency is a commonly highlighted issue. The recognition provided by Wii Remote has several restrictions, as previously discussed, making it more suitable for pointing and simple position or orientation tasks. However, the Wii Remote Plus and PS Move devices, combining accelerometers and gyroscopic technologies, are potentially the ones providing results closer to reality and without significant latency.

2.3 Data Collection

Having detailed the benefits and drawbacks of some of the most relevant gesture devices available in the market that enable to track movements, it is now important to understand what practical use they are intended to serve. Delivering a more user-centered framework by collecting motricity data, gives the chance to propose a set of exercises adapted to his greater limitations. Therefore, it is essential that the data being collected is accurate and relevant in order to achieve the stipulated goals, i.e., it should enable the system to take relevant conclusions about his physical condition.

Despite the relevance of appropriately selecting a gesture device to provide motricity data, it is also crucial to choose a framework that is able to interpret that data and turn in it into information readable by the developed framework. The next sections address further information about some of the most relevant libraries and frameworks, including the specifications, benefits and drawbacks.

2.3.1 OpenNI SDK

Intended for an easier software development for 3D motion sensing devices, like Microsoft Kinect and Asus Xtion Pro, the OpenNI framework is an open source development kit by PrimeSense that includes a set of APIs for voice, hand and body recognition, currently in version 2. The standalone framework is intended for C++ and Visual Studio development and enables cross-platform development between Windows, Linux and OSX [Opeb]. One of the aspects that greatly distinguish OpenNI from similar frameworks is the amount of third party middleware libraries that use the data provided by OpenNI to deliver specific practical applications, including tools and

wrappers, due to its open source SDKs. Also relevant is the growing community activity, team support and the online distribution channel. Follows some of the most relevant third party software [Opec].

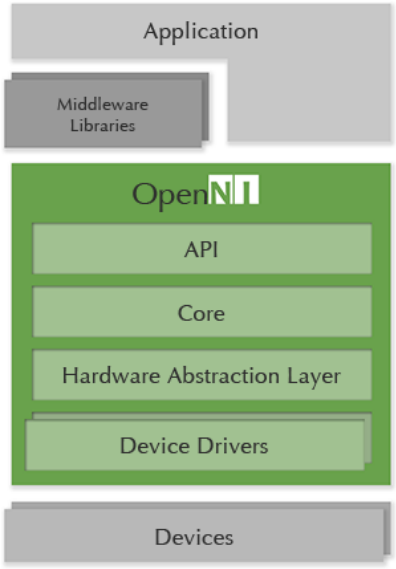

Figure 2.8: OpenNI2 architecture overview [Opeb]

- **NiTE** - developed by PrimeSense, NiTE is the most popular OpenNI middleware and includes a set of algorithms that enable both hand and body recognition for skeletal joint tracking for up to six persons simultaneously. It also features multi-platform development, including Android, and is used along with the OpenNI SDK by most developers to deliver additional high-level features [Pri].

- **SigmaNIL Framework** - features hand shape and gesture recognition and hand skeleton tracking and was developed by SigmaRD. It features a modular development, enabling developers to create and include new modules and, since it is an open source library, existing code can be modified in order to adapt the behaviour of existing modules [Sig].

- **GST API** - this Windows only middleware library was developed by GlobalSensing Technologies and uses NiTE algorithms to enable postures labelling, i.e., recognizes if a user is, for instance, standing, sat down, raising arms, among others [Glo].

- **ViiM SDK** - presents six main processing units (i.e. modules), which includes gestures recognition, user cropping, skeleton tracking, user segmentation, face analysis (under development) and optical flow analysis. Although it supports cross-platform development and all sensing devices, this development kit is paid and not open source [CoV].

- **ZDK for Unity3D** - Zigfu Development Kit is a middleware library that wraps OpenNI and NiTE frameworks with Unity, thus enabling 3D sensing for use with the popular game

engine. Unity featured programming languages include C# and Javascript, and enables additional deployment options at no additional cost, such as Android or web browser, the latter requiring the Unity Web Player plugin [Unia]. The ZDK package also provides a set of open source functional samples, HTML5 development, multiplatform support and an API that delivers additional high-level features [Ziga].

- **3D Hand Tracking Library** - the main feature of this library is the ability to track full hand articulation with up to 26 degrees of freedom, without requiring calibration. Due to its focus in hand tracking, it achieves a precision level most libraries lack. It supports both Windows and Linux development [FOR].

2.3.2 Move.me

Move.me is a server application that runs on the PS3 system. It performs the standard tracking and data collection of all the PlayStation Move controllers connected to the platform and makes it available over a local network to any computer asking for the data, which means a PS3 and a PC must be running in the same network for this process to work [Car]. It enables Windows development, with a well-established C# library, but also Linux using a C library, and supports Unity3D development [Car] [Mos11]. The development kit is available on the PlayStation Store for about 100 dollars, though Academica Program provides free access to students [Pla11]. However, being a library with restrict access, it has a negative impact in the amount of hackings and practical applications of the Move controller, with the community having little activity and lack of technical support, overall. All developed software must be approved and licensed by Sony before being distributed.

Figure 2.9: Move.me diagram overview [McC11]

2.3.3 Wii Remote libraries

Brian Peek's WiimoteLib is an open source library for Windows that supports Wii Remote and Wii Balance Board devices, and includes an API both written in C# and Visual Basic [Cha]. However, It was last updated in 2009, with MotionPlus accessory compatibility left incomplete and containing known issues, while the Wii Remote Plus device is totally unsupported [Cod]. In 2010, though, a library originally based upon the WiimotLib, WiiYourself! developed by gl.tter, received an update featuring MotionPlus fixes and enhancements [gl.]. Linux support is enabled by CWiid, a collection of tools written in C that includes a library with partial MotionPlus support and test applications [CWi]. Just like any other of the aforementioned libraries, though, CWiid development is stalled since 2010.

2.4 Conclusions and Related Work

The state of the art research addressed in this technical report, focused on exergames, NUI devices and frameworks for interacting with the gestures devices, provided useful information that enabled to take decisions with an increased awareness of the facts regarding the most favourable software and hardware technologies that would allow for a more advantageous development of a framework for assisting impaired children using natural interfaces.

Given all the possible choices analysed in this report, it was decided that combining the Kinect sensor with the ZigFu development kit, based upon OpenNI and NiTE frameworks, provides a set of unique and relevant benefits over any other possible choice. Besides allowing for a cost-effective and accurate approach, this combination:

- Enables full body tracking for a wider coverage of motor impairments;
- Does so without requiring body-attached hardware components [FPT12];
- Provides the opportunity to create AR environments and experiences with OpenNI specific methods [mJH12];
- Features an ever-growing set of free to use third party middleware libraries and APIs [Opec];
- Enables the Unity game engine to be used, as well as cross-platform development and multiple deployment options;
- Witnesses a continuous support by the development team for both the hardware and software technologies, as well as an active community willing to help.

On a final note, the Kinect sensor's latency is slightly higher than PS Move and Wii Remote Plus, though it should not significantly affect the design of exercises to be implemented in the framework and game prototype [dSMPL^{+}12], given that motor rehabilitation greatly relies upon slower paced exercises.

Chapter 3

Methodology

In order to deliver an NUI-based framework that would assist motor disabled children to improve or delay the progress of their limitations in the rehabilitation process, as covered in Chapter 1, it was essential to know what currently available technologies would better serve this purpose, as detailed in Chapter 2.

The development process adopted in this project context was a combination of the iterative design and incremental model methods. As the framework development was expected to have a considerable amount of requirements to be implemented, tested and evaluated, an iterative approach enabled to split the problem into smaller chunks, thus allowing a more focused and rigorous development during each iteration with fewer risks and more predictability during testing phases, as opposed to a single waterfall iteration.

However, the system specifications of this project had to be defined and established prior to the development cycle, since these are not provided by an external agent. Regarding this scenario, where the framework solution was not perfectly shaped before starting the iterative development, an incremental approach was considered to be of great relevance, given that a continuous and constant requirements changes was expected alongside the development.

An overview of the development process stages follows.

- **Technology exploration** - aimed at understanding how to take advantage of the selected technologies in the current context and what limitations to take into account;
- **Target definition and exercises design** - motor disabilities that could be addressed and what types of exercise suit the technological characteristics best, based on the information collected during the previous stage;
- **Framework development** - focused on the framework's architecture, design and implementation stages, along with the testing and evaluation phases on an iterative basis.

The first two stages were part of the initial planning phase that precedes the adopted iterative approach of the framework development. Therefore, the correct fulfilment of these two stages' objectives was of the utmost relevance in triggering a successful development. Nevertheless, the

obtained outcome is not meant to be definite and unchangeable, as the adopted incremental model is designed to consider emerging requirements expected from the learning experienced in earlier development iterations. The next sections address in more detail in what consisted each of the stages, which are then thoroughly detailed in upcoming chapters.

3.1 Technology Analysis

The first stage of the development process is detailed in Chapter 4, and consisted of exploring the hardware and software specifications, along with the limitations they carry and the need for calibration.

Regarding the hardware component, it was relevant to perform a set of functional tests to better understand how Kinect's constraints, aforementioned in the state of the art chapter, affect the way the device is intended to be used in this project's scope. Thus, it was important to undertake test suites that would address matters such as the detection latency, depth tracking, user height, required light and space conditions, or the sensor field of view practical implications.

Concerning the selected software, the functional tests conducted were intended to get acquainted with the development tools that would be part of the implementation process and understand how to take advantage of its features and avoid existing constraints.

3.2 Target Definition and Exercises Design

The information collected about the selected hardware and software, during the first stage of the whole process, enabled the second stage, detailed in Chapter 5, to focus on what kind of motor impairments would benefit the most from these technologies, thus defining the application's targeting. With the target in mind, and along with the previously gathered technological limitations, a series of exercises were then designed and tested with functional prototypes similar to the ones featured in the final framework.

Throughout this stage, a critical support was provided by an Occupational Therapy specialist, who actively assisted this project by providing useful information of existing paresis and plegia kinds of impairments, which ultimately led to define the test group, as well as a great insight on the most common exercises performed during rehabilitation sessions. Functional tests were conducted in order to validate which exercises could be accurately tracked by the Kinect sensor and to study possible changes in how they are designed to override known constraints.

3.3 Framework Development

As the information obtained in the first two stages of the process was gathered, the last stage consisted of developing the framework. Using the targeting and set of exercises proposed in the second stage as a case study, it helped guiding and testing the framework throughout the development stage. Each iteration of this stage comprehended the analysis, design and implementation

phases of the functional requirements, followed by an evaluation process mostly based upon functional testing. Adopting an incremental method enabled a more focused and rigorous testing phase throughout each iteration and to quickly identify possible errors found during regression testing, mainly because smaller changes were implemented each time.

Once the results obtained during the evaluation phase produced the expected outcome, a new iteration would start by revising the previously planned specifications and eventually perform amendments to the application's design and add new functional features. The iterative process was repeated until the established requirements were fulfilled, thus leading to the framework deployment and making it ready to be validated with the test group. The framework architecture and design are detailed in Chapter 6, and the implementation in Chapter 7.

Chapter 4

Technology Analysis

The study addressed in Chapter 2 regarding the subjects closely related with this project altogether with the adopted methodology, as described in Chapter 3, led to the need of analysing the hardware and software to be used in the application. This analysis is of great importance to the whole development process because the framework can only be accurately specified if there is a previous technological insight. The direction and success of the project may be influenced by that insight, given the relevance of understanding how the hardware and software components behave to certain scenarios and what constraints can be exploited when used in specific conditions.

The current chapter covers and details the hardware device used and the selected software tools and middleware frameworks, as well as the limitations that surfaced from the combination of these technologies. Functional scenario testing was one of the activities adopted during this stage and aimed at outlining conclusions that would make some decisions to be taken more cautiously during the second stage, as well as getting acquainted with the framework API, the development environment delivered by the IDE and the game engine's editor.

4.1 Hardware Device Specifications

The Kinect device used in this project was initially released by Microsoft in 2010 and was intended to be used with the Xbox 360 system, though early hacking and unofficial open source drivers enabled for domestic use with a computer [Per10]. Nevertheless, Microsoft eventually released a new hardware version of the device intended to be used on a Windows computer, along with an SDK to be exclusively used with that version of Kinect. Despite the lack of an official report regarding the matter, both versions of the Kinect device are believed to be virtually the same hardware-wise, though the Kinect for Windows official SDK contains additional features, such as API improvements or a near mode depth detection, and ongoing official support. Follows the Kinect sensor components and specifications:

- RGB camera capable of storing three channel data in a 1280x960 resolution to capture color image at 30 frames per second;

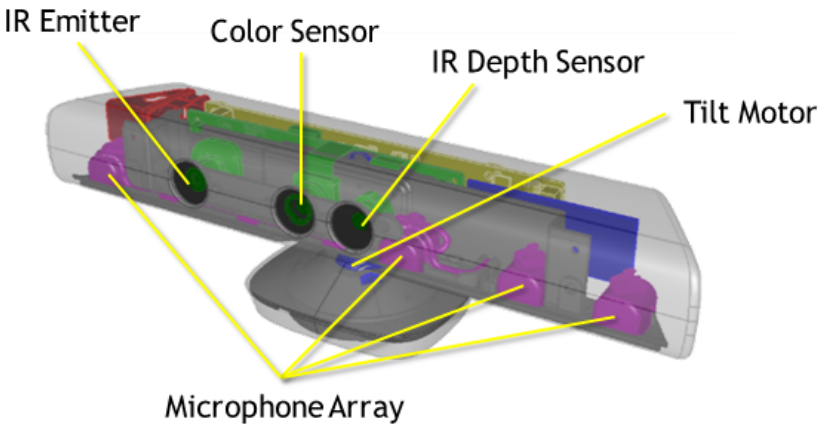

Figure 4.1: Kinect components overview [Kina]

- IR emitter that emits infrared light beams, and depth sensor intended to read the IR beams reflected back to the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor to enable depth image capturing. The depth map is computed using structured light with depth from focus and depth from stereo's parallax techniques;
- Multi-array microphone containing multiple microphones for capturing sound, enabling to locate the sound source and obtain the direction of the audio waves;
- 3-axis accelerometer to determine Kinect's current orientation;
- Viewing angle with a field of vision of approximately 43º vertical by 57º horizontal;
- Vertical tilt motor with a maximum range of 27º, whose angle elevation must be done using specific API functions.

However, this lightweight, small-sized and easy to carry device that delivers to a broad audience the first successful approach of full body motion controls free of body-attached components, has a handful of limitations that need to be addressed prior to specifying the framework. Regarding this thesis scope, the current section addresses detailed information on some of the main constraints. During this step, some of the samples included with the official SDK were used.

4.1.1 Joints tracking and accuracy

Using a common Kinect development kit, the device enables up to twenty joints to be tracked simultaneously, as illustrated in figure 4.2. While this proves to be a reasonable amount of joints for exercises based on gross motor skills, fine motor skills usually require the coordination of small muscles of specific body parts, such as the fingers, which the Kinect is unable to track without relinquishing full body tracking. This trade off emphasizes that Kinect specifications are

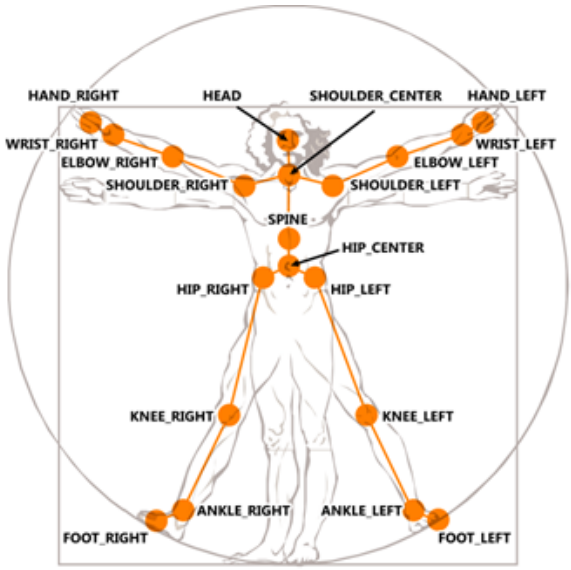

Figure 4.2: Joints tracked by Kinect [Mic12]

not designed to address 1:1 precision throughout the whole body joints, but a simplified set of joints tracked with a satisfactory precision. Figure 4.3 illustrates the skeletal tracking feature.

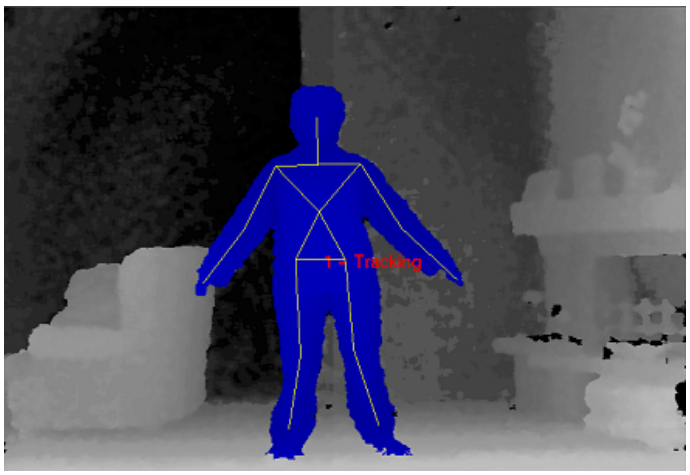

Figure 4.3: Skeletal tracking sample using OpenNI framework

Regardless of the amount of tracked joints, one of the most significant concerns when designing applications based on Kinect is understanding how accurately it can track these joints. Although Kinect's capabilities have proven to be effective in gross motor skill tracking most of the cases, small levels of skeletal jittering can be detected; even if certain algorithmic functions can correct this undesired behaviour by artificially stabilizing the obtained values, fine motor exercises will not take advantage of accurate results. This is mainly due to the depth sensor's resulting depth map not providing enough depth precision to enable computer vision algorithms to have greater performance.

Another limitation commonly pointed out is the lack of tracking precision when a set of objects overlap, i.e. when an object gets between the camera and another object. One common example of this event occurs when one arm passes in front of the human torso and to the other side of the body, which may result in the Kinect depth sensor lacking relevant data about the overlapped joints and thus decreasing the tracking precision. The consequences may vary depending on how close the overlapping body parts are to one another, as well as the Kinect camera positioning and orientation in respect to the user, or the guessing behaviour of the computer vision algorithm when dealing with this kind of situations. This imposes some restrictions to the plethora of exercises that can be performed and requires special regard when determining what activities to include in the framework prototype. Figure 4.4 shows a skeletal tracking miscalculation: both arms are together to the left side of the body, though the skeleton draws the left arm vertically.

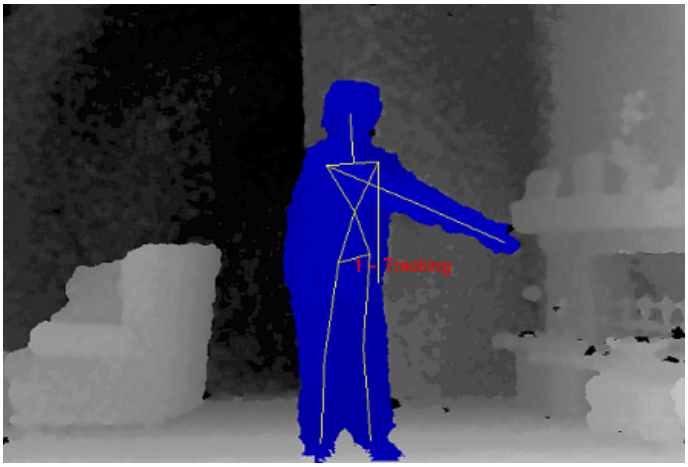

Figure 4.4: Arm overlapping the torso

Joints orientation is another subject that requires particular attention, as data provided by the depth sensor may not be enough to ensure the feasibility of all possible human body rotations. Therefore, it is relevant to bear in mind that a real-time, human shaped, virtual model is not expected to precisely mimic all human body rotations, and thus exercises heavily relying on joints orientation may have to deal with inaccurate data.

4.1.2 Field of view

As previously detailed, the Kinect IR camera has an FOV of 57º horizontally by 43º vertically. While these values are on par with low to mid range webcams, full body tracking is often necessary when designing motor based exercises and may require the user to keep a significant distance from the Kinect, depending on his height. The required distance may have to be greater when considering that raising the arms above the head asks for additional vertical focal length. The lack of field of view beyond a standard value may constrain the amount of realizable types of exercise and cause frustration if certain body parts are out of the field of view at some point during the exercise execution. However, given that the application is addressed to children, usually lower

than the average adult, the risks of vertical field of view causing a negative impact on the exercise design process are smaller. Figure 4.5 shows a conducted field of view test with a 1.62 meters tall adult; with arms down, the required distance from the Kinect sensor was 1.97 meters; with the arms raised, the distance was approximately of 2.66 meters. This test indicates that a standard sized young child would require about 2 meters from the Kinect sensor.

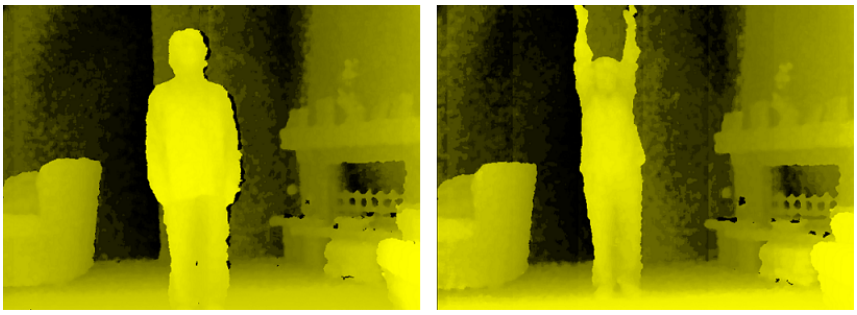

Figure 4.5: Field of view test with both arms down and raised

4.1.3 Depth range

Besides the field of view, the depth range is another important restriction to consider when designing exercises aimed at Kinect. Figure 4.6 illustrates the range of depth (in meters) the Kinect sensor is capable of detecting, with the teal coloured zone featuring the depth range where the best tracking results are expected.

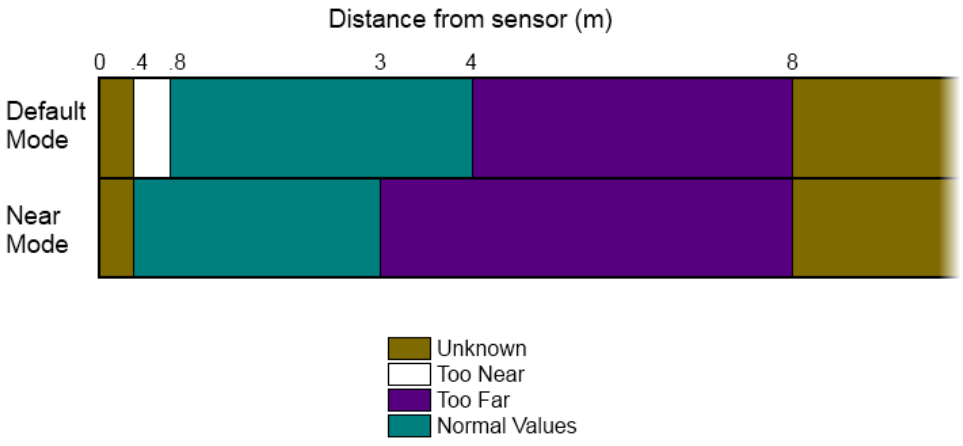

Figure 4.6: Kinect's depth range in default and near modes [Kin12]

The FOV minimum distance test complements the 4 meters maximum limit to determine the in depth movement available to a mid-sized child consists of almost 2 meters. This provides more than enough tracking area space for fixed-position exercises and even exercises requiring the user to change position throughout the exercise.

4.1.4 Environmental Conditions and Sensor Placement

Kinect Sensor's captured images and range measurements may be affected by certain variations in the environmental conditions. Microsoft has not previously provided detailed reports regarding certain environmental influences pointed out in successive independent studies, such as thermal and air draft variations. According to [FM12], both RGB and IR cameras present greater blurriness levels and loss of sharpness when higher temperature of the internal components is experienced. This increasing thermal condition as well as air drafting have a reasonable negative impact in the range measurement obtained by the depth sensor and thus should be avoided to prevent further computational errors.

Another matter that is widely regarded as having considerable influence on the depth sensor computed depth image is the lighting conditions. Since Kinect depth sensor reads IR rays emitted back to the camera, the device becomes unable to be correctly used under direct sunlight, since the infrared rays emitted by the sun highly affect the depth measurements. Apart from making it unbearable to use Kinect outdoors during the day, room windows may have to be closed in order to obtain adequate results, given that IR rays may be reflected in certain objects contained in the camera field of view, and back to the sensor, thus negatively impacting measurements. Artificial lights do not emit infrared radiation and have no known impact on Kinect's performance. Figure 4.7 shows the depth image on the left and the RGB camera image on the right; due to the sun's reflected infrared rays, the depth image draws black in that area.

Figure 4.7: Kinect's depth sensor behaviour to sunlight conditions

Sensor placement is also relevant in order to prevent any undesired objects from obstructing the viewing angle, such as the surface where Kinect is placed on. It is recommended to horizontally center the sensor with the user's initial position, prevent reflective surfaces from being placed ahead of the device and any mirrored surface within the viewing angle [Kinc].

4.2 Software Specifications

The current section focus on the selected software for developing the framework. It details Unity and ZDK main specifications and covers test suites that were conducted in order to exploit existing limitations.

4.2.1 Unity Game Engine

Unity3D is a widely used game engine that enables cross-platform development developed by Unity Technologies and currently on version 4.1. Some of Unity's most interesting features include a substantial amount of ready-made components, such as rendering, physics, colliders or controls, debugging support that shows variables tweaking over time and enables these variables (and even the scripts) to be changed while running the application, an integrated, a complex yet comprehensive level editor UI, a wide range of supported assets formats, the ability to write code in Javascript, C# or Boo, deployment to multiple desktop, web and mobile platforms [Unib]. All these tools highly assist the developer in increasing the development speed of the projects. It relies on MonoDevelop IDE for activities such as code writing and additional debugging and compiling settings.

Figure 4.8: Unity Editor [Unic]

4.2.2 ZigFu Development Kit

ZDK is a framework that enables cross-platform development of Kinect based applications for Unity, HTML5 and Flash. This allows the developer to easily deploy an application to multiple platforms. The ZigFu middleware framework requires OpenNI version 1 or 2 and NiTE frameworks, or alternatively the Windows SDK if a Kinect for Windows device was being used, to be installed in the computer in order to work properly [Zigb]. This is the case because ZDK is focused upon delivering a more comprehensive and complete API based upon these development kits in order to provide a set of additional features and an accessible Unity/Kinect wrapper.

Some of the novel features ZigFu library provides include the ability to detect certain hand gesture patterns such as push, directional swipe and wave. It also includes a custom update function called every frame that provides full body skeleton data of the tracked user for that frame, including joints position and orientation and the user current position in depth, breadth and height.

Other relevant features include events that trigger when a user enters or leaves the scene, whether the sensor is currently connected, and control variables for the number of users being tracked during a frame. However, and besides not being open source, the API documentation does not provide detailed information regarding the featured classes [Zig12], thus making it harder to take total advantage of ZigFu's capabilities. On the other hand, the Unity package made available by the ZigFu team comprises the framework and a set of sample scenes using the provided API functions. Each one of these samples is intended to demonstrate practical examples of what ZDK has to offer, with AvatarFrontFacing (AFF) being the most relevant in regard to this project's scope.

The AFF sample features a human shaped 3D model in a plain three dimensional environment; in line with the 20-joint limit, the avatar mimics the user movements as if it was a mirrored image, and computes real time joint rotations and body position in the 3D space. A top view radar in the upper right corner of the screen shows the user current position in the whole area covered by the depth sensor, while the bottom right corner displays the depth image. For the mimicking feature to be triggered, though, the user body must first be detected by the framework, due to calibration purposes of the version 1 of the OpenNI framework. This calibration is performed by assuming a T pose with the body, i.e. with arms in an outstretched horizontal position, as indicated in figure 4.9.

Figure 4.9: The required T pose detection

In some cases, joints rotation is performed based on the orientation of other joints, in order to prevent inaccurate tracking results. This has been experienced with the head rotation, which is always on par with the shoulders' orientation, preventing the avatar from rotating the head joint independently of the shoulders joints, thus resulting in an avatar constantly "looking forward" and unable to turn the head in a certain direction, as illustrated in figure 4.10. Another limitation is the inability to effectively detect some arm rotations, as shown in figure 4.11.

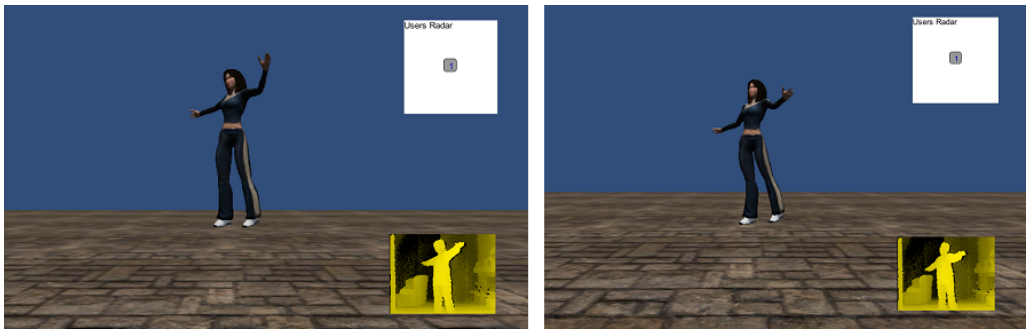

Figure 4.10: Example of joint orientation issue regarding the head rotation

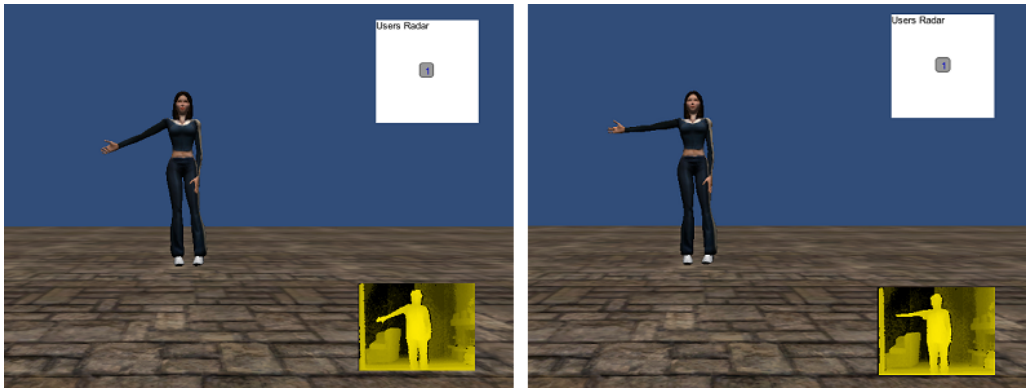

Figure 4.11: Example of joint orientation issue regarding the right arm rotation

Chapter 5

Target Definition and Exercises Design

In Chapter 4 a detailed specification of the selected technologies was presented, with a strong focus in exploiting their limitations, mostly supported by performing test suites, along with research in some cases. The outcome was enlightening and served as a guideline to better delineate the next stage of the development process, that consisted of defining the framework's target from the whole spectrum of existing motor disabilities, as well as the set of exercises to include in the prototype and validate the application.

An adequate approach to achieve this stage goals required a significant knowledge in fields of study that dealt directly with motor and cognitive impairments, such as physiotherapy, psychology and occupational therapy. This triggered the need to establish contact with researchers in the aforementioned scientific areas in order to obtain assistance in making decisions regarding the motor impairments to include in the study and what exercises were a standard in the rehabilitation process of children with the selected disabilities. Eventually, the researcher and specialist in Occupational Therapy Professor Joaquim Faias from Escola Superior de Tecnologia da Saúde do Porto (College of Health Technology of Porto) was reached, with previous experience in technological devices applied to children cognitive development. Professor Faias was closely and actively involved throughout this whole stage of the development process, as well as during the validation process, assisting the team with great insight in the rehabilitation field every time it was required.

5.1 Targeting

The first step consisted of defining the motor impairments that were consistent with and fit to the technological characteristics and project goals. Disabilities dealing with total paralysis of certain body parts seemed immediately out of consideration, as the technology choice did not take into consideration children that required to have close attention and continuous assistance by a professional. Professor Faias was aware of the Kinect sensor capabilities prior to the established contact, which eased the communication, so the role of the team consisted of referring to the limitations collected in the previous stage. After having acknowledged of the constraints, Professor Faias suggested spastic diplegia and hemiparesis as two impairments that could highly benefit of the

motion tracking technology and fit into its limitations. What motivated this decision was the ability of these children to interpret the movements of a mirrored avatar to some extent - depending on the cognitive condition of the child; to be able to perform full body, motricity-based exercises, even if some difficulties are experienced depending on the severity of their limitation; and do not require to be physically assisted while performing the exercise. Professor Faias selected children included in Levels I and II and aged between 6 and 12 years old as the most appropriate group test in this context, according to the "Gross Motor Function Classification System - Expanded and Revised" [PRBL07], which were described as follows:

> Level I: Children walk at home, school, outdoors, and in the community. Children are able to walk up and down curbs without physical assistance and stairs without the use of a railing. Children perform gross motor skills such as running and jumping but speed, balance, and coordination are limited. Children may participate in physical activities and sports depending on personal choices and environmental factors.
>
> Level II: Children walk in most settings. Children may experience difficulty walking long distances and balancing on uneven terrain, inclines, in crowded areas, confined spaces or when carrying objects. Children walk up and down stairs holding onto a railing or with physical assistance if there is no railing. Outdoors and in the community, children may walk with physical assistance, a hand-held mobility device, or use wheeled mobility when travelling long distances. Children have at best only minimal ability to perform gross motor skills such as running and jumping. Limitations in performance of gross motor skills may necessitate adaptations to enable participation in physical activities and sports.
>
> *Gross Motor Function Classification System - Expanded and Revised, page 4*

Follows a more thorough explanation of each of the aforementioned impairments.

5.1.1 Spastic diplegia

Cerebral palsy spastic diplegia is a chronic neuromuscular condition of hypertonia and spasticity most prominent in the muscles of the lower extremities of the human body, though arms and face may also show signs of muscle stiffness, with tendon reflexes being typically hyperactive [Nat12]. Spastic diplegic children may experience varying degrees of severity, from barely noticeable variation in the movement to extremely pronounced balance and gait motor disorder. This impairment is widely associated with scissor gait pattern, a gait abnormality featuring flexion of the legs at the hips and knees, and also knees or thighs crossing over, thus resulting in a walking similar to a scissored gait [RG01]. In lower severity degrees this pattern may not be very pronounced or almost unnoticeable. Most cases of spastic diplegia are caused by a trauma while the baby is still in the womb, and a few due to head injury or infection after birth; prematurity or low birthweight may also be the cause of this impairment [Nat12]. The most common treatment that can be used for all spastic diplegic children is the physical therapy, which aims at preventing muscles from becoming unable to move due to disuse [Har].

5.1.2 Hemiparesis

Hemiparesis, also known as spastic hemiplegia, is a one-sided weakness of the body and the most common movement impairment [Nat12]. It may affect the legs, arms, hands and even facial muscles, and some of its symptoms include loss of balance, and lack of coordination and movement precision, depending on the severity degree [Wei10]. However, there is a particularity with hemiparesis which is related with the location of the existing injury in the brain. Besides affecting the motor ability, an injury on the left hemisphere of the brain (i.e., right-sided hemiparesis) may affect the child's ability to both speak and comprehend language; on the other hand, children with a right-sided hemisphere injury (i.e., left-sided hemiparesis) may denote lack of memory and attention span, as well as to speak excessively [Wei10]. As with spastic diplegia, the treatment of hemiparesis relies largely upon rehabilitation with physiatrists and occupational therapists [Wei10].

5.2 Types of Exercise

Patients affected with any of the forms of cerebral palsy described in Section 5.1 can improve their physical condition by performing exercises with a specific focus, usually at occupational therapy sessions assisted by professionals. Motor planning and coordination are some of the abilities the exercises are aimed to address.

This section addresses a set of exercises suggested by Professor Faias, upon presentation of the technological context the application would be implemented in, and considered to be of great relevance in the motor development of patients featuring one of the previously detailed disorders. The description of each exercise specifies how the child is expected to perform it, i.e. the initial, intermediate and final positions, goals intended with its execution, whether the tests performed with the selected technological context led to a positive outcome and, if this is the case, possible adaptations needed to take full advantage of the device specifications.

The test suites were performed using a functional prototype featuring the selected hardware and software technologies, applied in a context similar to the intended with the final prototype, and had the collaboration of Professor Faias supervising the obtained results. The tests were executed running a scene on Unity based upon the AvatarFrontFacing sample scene included with the ZigFu Development Kit and detailed in Section 4.2.2. The performance tests featured a person with no motor or cognitive impairments and the evaluation of the feasibility of the exercise consisted in observing how similar the mirrored avatar behaviour was to the original user performance. Follows a description of the exercises design and test results.

5.2.1 Schilder test

The starting position for this exercise requires the patient to front raise both arms along the sagittal plane, keeping them in a horizontal position, 90^{o} angle with the body and parallel to one another, with palms turned down. The remaining body parts are kept in the standard anatomical position.

An alternate set up, yet with similar functional results, requires the patient to perform the arms abduction along the coronal plane, i.e. raising the arms laterally away from the median sagittal plane of the body. In order to perform the exercise, the patient is required to alternately rotate the neck to one side and the other slowly, i.e. turning left and right, repeating the task a predefined number of times. An alternate way to perform this exercise is to laterally move the head away from the midline of the body and toward the shoulder, i.e. lateral flexion or neck abduction. The goal of this exercise is to understand the ability of the patient to keep the arms rose (i.e. in the starting position) while performing the head rotation.

Due to Kinect's lack of precision in effectively tracking fine motor skills, such as neck rotation or neck abduction, the results obtained during the tests did not meet minimum requirements needed to take significant conclusions on the patient's condition, thus leading to unsatisfactory results. On the other hand, and since the focus of this exercise is to monitor whether the patient was able to keep the raised arms as stable as possible, an alternate setting was to consider the intervention of an occupational Therapist to ensure that the child performed the neck abduction/rotation task as intended. However, this would counter one of the main motivations of this study: the chance to provide a more autonomous motor rehabilitation process. Figure 5.1 reflects the lack of head rotation tracking feature.

Figure 5.1: Schilder test

5.2.2 Raising the arms above the head

The initial position of this exercise is similar to the standard anatomical position, with the palms facing forward or backwards. During the execution, the patient is required to raise the arms above the head along the sagittal plane, until the upper arms are slightly above the horizontal position, forearms are vertical and the hands facing up, thus the arms are not intended to be fully stretched. Following this, the reverse movement is performed until the patient reaches the initial anatomical position, repeating the task three times. This exercise evaluates the child's ability to raise and lower both arms while keeping the movement coordinated and provides useful data on the expected offset between left and right arms' performance.

Regarding the obtained test results, all steps were effectively tracked by the depth sensor and mapped to the avatar, thus no technological limitations were met. Figure 5.2 shows an accurate detection of the upper limbs during execution of the exercise.

Figure 5.2: Raising arms test

5.2.3 Sequence of poses

This exercise requires the child to be able to interpret and mimic a pose. To achieve that, it was important to deliver an exercise that consisted of performing a sequence of poses, while providing the child the time needed to mirror it as correctly as possible. The goal of this fragmented exercise differs from an exercise that demands a continuous movement because, in this case, it is relevant that the child is able to assume each of the designed poses, ignoring how he performed the whole movement but considering how much time he required to achieve it.

There are a multitude of poses that meet this exercise goals, though it was recommended to feature no more than four of them in the whole exercise. Figure 5.3 illustrates the set of suggested poses along with the pose mirrored by the avatar. As the pictures evidence, poses 1 to 4 were successfully tracked and mimicked by the avatar. Poses 5 and 6 had overlapping body parts allied to a more complex posture, and thus have shown inaccurate results.

5.2.4 Grabbing ball thrown by an external agent

This exercise required the intervention of an external agent in order to be fully performed. It consisted of having the external agent, e.g. the therapist, throwing a ball in the direction of the child and understanding his ability to successfully grab it. While this can be easily monitored in a standard physical rehabilitation scenario, there is a heavy unpredictability factor in the ball

Figure 5.3: Poses test

trajectory, caused by the agent responsible in throwing the ball, that makes it infeasible to apply in a context where the patient is required to mimic a previously recorded exercise.

5.2.5 Bouncing a ball

In order to perform this exercise it was required that the patient bounced the ball to the ground a number of times. As with the previously described exercise, the unpredictability of the ball trajectory led to unwanted results in comparing two executions of the same exercise, thus possibly providing the framework with misleading results that do not translate the real capabilities of the patient in achieving the exercise required goals.

5.2.6 Side steps

The side steps exercise requires the patient to start in the anatomical position, front facing the Kinect as centered as possible. It requests the child to perform side steps in each direction: two side steps in one direction (e.g. left), then two side steps back to the center, two side steps to the opposite direction, and again back to the center of the visible space. The main objective of this exercise is to monitor whether the child is capable of performing side steps with the correct leg abduction and without flexion of the knees or hips. The test results demonstrated a satisfactory accuracy in tracking the joints of both the hips and knees, as shown in figure 5.4.

5.2.7 Scissors jump

The scissors jump exercise starts in the anatomical position. It requires the patient to perform a jump where both legs and arms are abducted, followed by a reverse movement, i.e. arms and

Figure 5.4: Side steps test

legs adduction, in such a way that the child returns to the initially adopted anatomical position. From the anatomical position, the limbs abduction is expected to be of approximately 120º arms and 45º legs. Even though this exercise is fast-paced and harder to execute in comparison to the other ones, it was considered to be of great relevance for the framework to feature it for the sake of having a greater variety in difficulty, in order to target children with lower severity degree of the impairment and obtain more significant results. The avatar accurately mirrored the performed movements and with no considerable latency, as can be seen in figure 5.5.

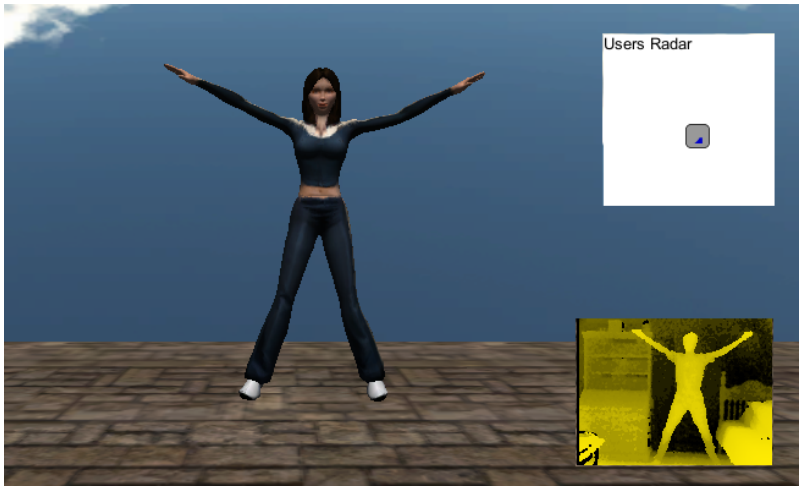

Figure 5.5: Scissors test

5.2.8 Conclusions

The previously detailed exercises were considered to be some of the most significant in the rehabilitation of individuals featured in the target-audience. However, and despite having been carefully selected according to the technological context, some of them were eventually dropped out, as they did not fully suit the technologies constraints. Therefore, the exercises chosen to be featured in the framework prototype were:

- Raising the arms above the head;
- Sequence of poses (poses 1 to 4 of figure 5.3);
- Side steps;
- Scissors jump.

Chapter 6

Framework Architecture and Design

Chapters 4 and 5 provided a detailed study of the technological environment, the group of impaired children that would benefit the most and the set of physical exercises with greater relevance. This chapter is focused in describing the adopted architecture component, as well as how it was designed, comprising the mechanics, play modes and data storage.

When defining the framework architecture, one of the main concerns laid in providing a set of mechanics that enabled any kind of motor-based exercise to be easily created and executed any time, while attending to the constraints previously gathered. Other relevant requirements included the ability to collect data of the user based upon the performance of executed exercises, as well as ensuring the depth sensor accuracy in tracking the user. An architectural representation of the framework is presented in figure 6.2, which features a components diagram approach. The framework is not responsible for the "Sensor Connection" and "User Tracking" components, though they are closely related with the framework components and delivered by the technologies featured in figure 6.1.

- Exercise creation - comprises two other components and is responsible for enabling the user to create a new exercise. The *exercise recording* component is responsible for mapping to

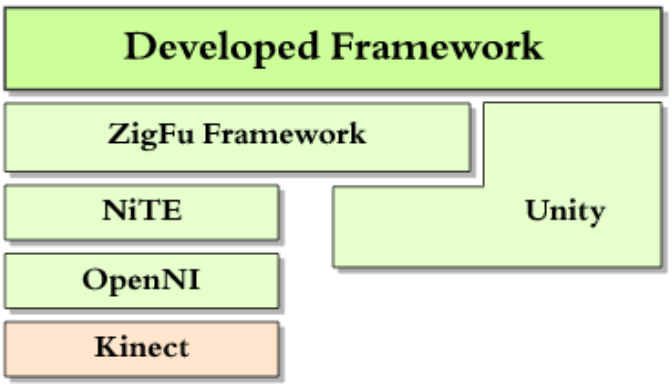

Figure 6.1: Block diagram overview

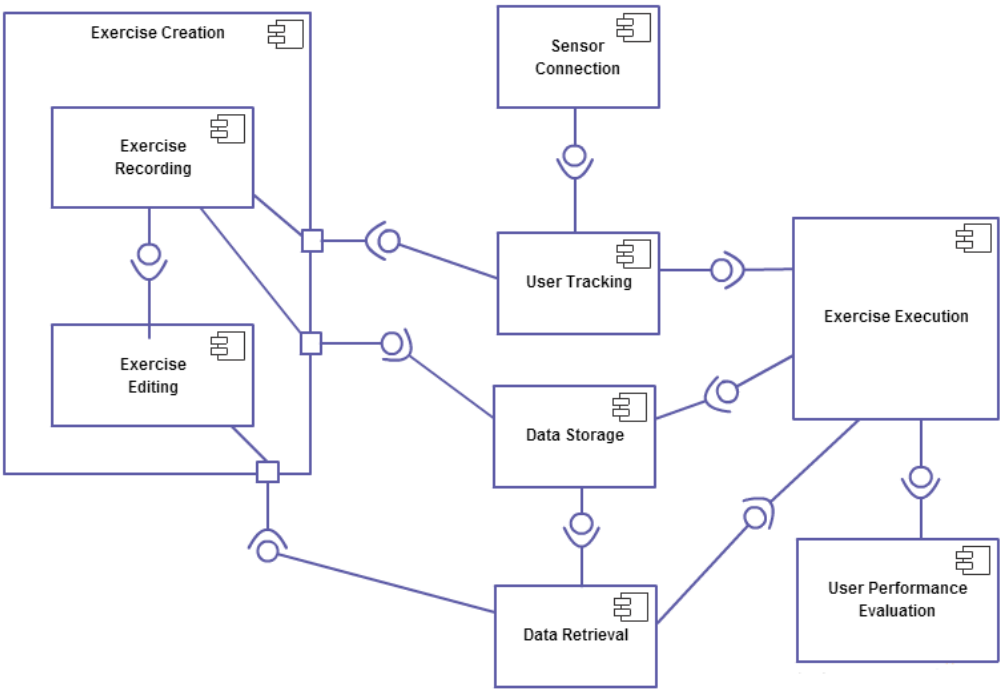

Figure 6.2: Components diagram overview

a virtual avatar the user tracked data provided by the *user tracking* component. *Exercise editing* enables the user to define additional settings of the evaluation process.

- Exercise execution - component responsible enabling the patient to perform an existing exercise. It requires the original exercise recording data and provides the data of the patient resulting data.

- Sensor connection - establishes the link between the sensor (hardware device) and the software components. Provides all the data captured by the sensor cameras and whether the sensor is connected and working properly.

- User tracking - component that interprets the depth image data provided by the *sensor connection* component and translates it into user data, such as the position and rotation of each of the body joints.

- User performance evaluation - uses the patient execution data from the *exercise execution* along with the exercise data to deliver data regarding his performance evaluation.

- Data storage - component that receives data and stores it for later use by *data retrieval* component.

- Data retrieval - provides data previously stored by *data storage* component.

6.1 Mechanics

This section details the mechanics featured in the framework and what factors motivated their addition. Explaining the mechanics leads to an easier understanding of the featured play modes detailed in the following section. These mechanics are an extension to the framework architecture presented and are intended to clarify how the framework actually works.

6.1.1 Execution Modes

The previous chapter referred to a set of exercises that were relevant to include in the framework, though some of them demanded a specific approach in order to be delivered. The objective of some of these exercises demanded that the child performed an uninterrupted and continuous movement, where it made sense as a whole. However, the sequence of poses exercise had a different approach, since it was relevant that the user would endeavour to perform each of the poses featured in the exercise and have the time to improve to their own rhythm.

Taking this into consideration, two distinct ways of designing the execution of an exercise were proposed:

- Continuous execution - intended for exercises that were designed to be executed uninterruptedly and respecting a specific pace, with patients being demanded to perform the exercise continuously within the imposed execution times, as the outcome would only make sense if the timings were fully respected. The exercises based on this approach are *side steps*, *raising the arms above the head* and *scissors jump*.

- Paused execution - this execution mode is relevant for exercises that are "fragmented" in a set of independent poses and where the movement between these is not relevant. Since the goal this time around is to perform static poses, and not a flowing movement, a different approach was designed that enabled the exercise to pause its execution in certain critical moments, thus providing the child with time to execute and improve the required pose. *Sequence of poses* is the example of an exercise that relies upon this execution mode.

6.1.2 Evaluation on Keyframes

Despite the decision of including two distinct ways of designing the course of an exercise, it was still required to come up with a mechanic that enabled to define the moments throughout the exercise where the child's performance was being evaluated.

- For exercises relying on a continuous execution an ongoing evaluation was first considered, as it was the simplest and more obvious approach. It consisted of matching the original performance with the child's performance for each frame. This was later dropped due to ignoring the fact that, even for an uninterrupted exercise, there were certain poses in the whole movement that stand out as being the key poses that the child should be able to achieve, and thus the performance evaluation process shifted to focus on these specific set

of moments. This made clearer that defining keyframes throughout the exercise would better fulfil this requirement.

- For paused exercises however, the aforementioned keyframe approach was taken into consideration right from the beginning and eventually adopted. This was mainly due to the fragmented and static nature of the poses featured in the exercise, which eased the decision of an evaluation based upon selected frames. The keyframes are the moments in the exercise where the avatar stands still, motionless and waiting for the child to be able to perform it.

6.1.3 Evaluation Metrics

The previous section focused on deciding when the evaluation would occur during the exercise. Thereafter, it was mandatory to answer the question on how the comparison between the original and the patient performances should be done, i.e., what metrics to take into account in evaluating the child's performance. Being a framework based upon mimicking the movements of an avatar, the first approach was to obtain, for each joint, the angle resulting from the joint rotation of the original execution and the joint rotation of the patient execution. The angle value, in degrees, is as smaller as closer the patient is from achieving each of the intended joint rotations. This represents the main process of evaluating the child performance and is, in fact, applied to both the continuous and paused executions.

However, there are some functional differences of the angles among execution types. While the performance evaluation of the continuous execution solely relies upon the resulting angles, for the paused execution it represents a way to monitor whether the child was able to effectively perform the pose or not; the elapsed time since the pose first emerged until the moment the child achieved it, stands for an additional evaluation parameter of the patient's performance. This comprises the data the framework collects about the patient.

To complement this, some additional rules were defined:

- **Weighted joints** - for each keyframe, it is possible to define the weight of a each of the joints, mostly useful for calculating average performance based upon that keyframe or throughout the whole exercise. It is expressed in integer value numbers, thus the higher the integer the greater the weight that joint will have. This also enables some joints to be ignored (attributing a joint weight equal to 0) in case they are not relevant in that keyframe, e.g., if it is asked that the child raises the arms, the legs' joints can be ignored as they have no influence upon what is being requested.

- **Angles threshold** - for each joint of every keyframe, an angle offset can be defined, known as the threshold, which corresponds to the maximum acceptable angle resulting from the original and performed joint rotations. Besides providing a better control regarding how well the child is performing, it is also an essential control variable in paused exercises, in that the motionless avatar (during a keyframe/pose) will trigger and proceed the movement

when the resulting angles of all weighted joints (i.e., joints whose weight is greater than 0) are less than or equal to the respective angle threshold value.

- **Comparison timeout** - in order to prevent frustration for not being able to complete a certain pose keyframe in a paused exercise, a timeout variable was added to prevent the patient from being indefinitely stuck in the same moment of the exercise. This timeout value, in seconds, represents the required time for the avatar to resume the exercise in case the patient is not being able to perform all angles of the pose according to the defined thresholds; if this occurs, the angle results stored correspond to the frame where the patient was able to perform it more in line with the original, based on the calculated weighted average angle, throughout the whole time span of the keyframe.

- **Frames threshold** - this behaves similarly to the *comparison timeout*, in that it is designed to prevent frustration on the patient. However, this is exclusively featured in continuous exercises and intervenes differently. During continuous exercises, and as a result of experimentation, is was observed that a user performs the exercise with a little delay in respect to the original performance. In order to prevent prejudicing an eventually significant yet slightly delayed execution, a frames threshold was created to enable the keyframe joint values to be iteratively compared with all the joint values of a given amount of frames of the child's execution following that moment. That is to say that it compares not only the exact same moment (i.e., the keyframe moment), but also an x number of frames after that, with x given by the frames threshold variable. As in the *comparison timeout*, the best weighted average angle, resulting from the comparison of the keyframe with all the spanned frames, is stored.

6.2 Play Modes

The framework interface enables the user to choose from a list of existing profiles; this profiling decision was introduced so the data of the executed exercise could be associated to each patient. It is then possible to select one of three existing play modes: create a new exercise, load an existing one or freely run the tracking scene without storing any collected data or retrieving existing data.

6.2.1 Create Exercise

The component dealing with the creation of a new exercise is mainly intended to add an exercise to the framework and comprises two steps: record and edit.

When the record step starts, the screen features a single avatar, that will mimic all the movements performed by the user, so the Kinect must be connected or otherwise. The HUD consists of the user radar and the depth map image, both using real time user data, as illustrated in figure 6.3. Throughout each frame of the whole execution, user data is being stored in a local file with all the required information to enable the framework to load it back correctly, i.e., the rotation and position of each of the active joints, as well as the user current position in the space visible, is

being stored. In case the user leaves the visible space of the Kinect sensor, the data will stop being collected, restarting by the time the user returns to the visible space. Clicking the "Finish" button located in the upper left corner of the screen stops the execution and no further user data is stored; this ends the first step of the exercise creation process and launches the edit scene.

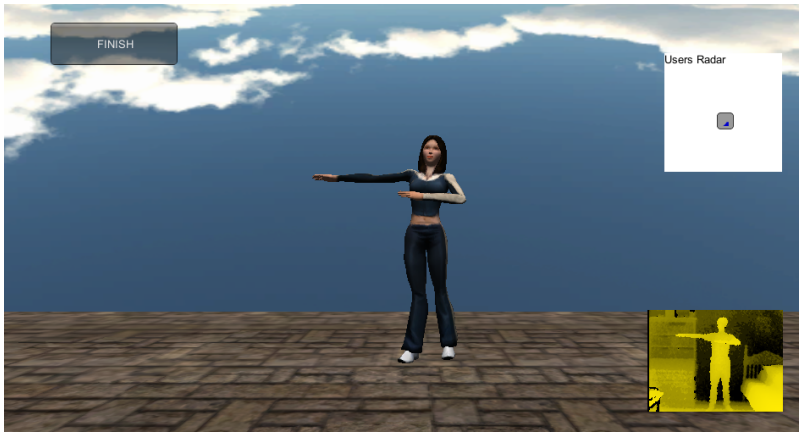

Figure 6.3: Exercise recording

The second step consists of defining the properties for the recorded exercise and the type of execution. In this scene the Kinect sensor is not used in any way, thus the HUD does not include the top view user radar and the depth map image, unlike the previous step, as illustrated in figure 6.4. The avatar will perform the exercise previously recorded in the exact same way by retrieving the data in the previous step. In the bottom of the screen, there is a set of interactive buttons; the "Play" or "Pause" button alternates a flag that will trigger the avatar mimicking behaviour, or else make it remain motionless in the same frame that it was executing when the button was pressed, respectively. This enables the user to define the keyframe values using the panel in the right side of the screen. A checkbox in the top is used to determine whether the exercise is intended for a paused or continuous execution; this is a variable defined for the whole exercise and is not related with the properties defined for each frame. Follows a list of the active joints along with two columns on the right, ready for data input of integer type values. The first column is intended for the angle threshold value, in degrees; the second column is the weight of the joint in the evaluation of the frame. The "Add" button in the bottom of the screen stores the data currently defined in the columns, along with the corresponding frame number, and resets the values. The "Play" button triggers once again the execution; if the avatar has finished performing the exercise, it will seamlessly restart executing it from the beginning, though no data is lost in the process; the frames counter in the bottom may assist the user in understanding whether the exercise is over or not. The "Finish" button ends this step and the whole process of creating a new exercise, thus it should be used by the time the user has finished adding all intended keyframes.

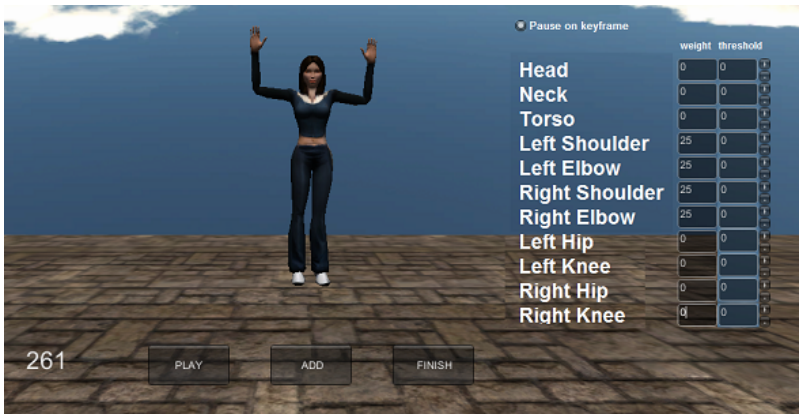

Figure 6.4: Exercise editing

6.2.2 Load Exercise

This play mode is intended for performing a previously recorded exercise. The most evident visual differences from the exercise creation bare the two avatars positioned side by side: the one in the left of the screen mimics the user movements and the one in the right, with a different coloured training suit, executes the originally recorded exercise, which is loaded by retrieving the motion data stored when it was first created. After being detected, the original avatar automatically starts performing the exercise. The patient is supposed to follow the movements as closely as possible. When a keyframe occurs, one of two things may succeed, depending on the execution mode:

- in case the exercise is **paused**, the avatar will remain motionless in the pose defined by the keyframe until the user performs all weighted joints according to the established angle thresholds; the framework will then store the resulting angle values and total elapsed time required to achieve that goal. If the child is unable to perform it within a time limit, the angle values stored are the ones originated from the frame that had the best weighted average angle, along with the time limit. When the exercise finishes, the average elapsed time of all keyframes is presented; for validation purposes, the angles results and elapsed times are separately stored for each keyframe.

Figure 6.5: Paused execution

- if the exercise is **continuous**, there will be no interruption of its execution, so it is relevant that the child is capable of performing it steadily and within a time frame. In order to ease the exercise complexity and not prejudice the child for not performing it with the expected simultaneity, a frame threshold enables the comparison of both executions with the best obtained weighted average angle, thus storing the angle values of the frame where this succeeded. When the exercise finishes, only the average angle of each joint and for all keyframes is presented, though all angles values are separately stored for each keyframe.

Figure 6.6: Continuous execution

6.2.3 Free Mode

This play mode is similar to the first step of the exercise creation mode in that an avatar mimics the user movements, thus requiring the Kinect sensor to be connected; it also features the top-view user radar and depth image. The only difference is that it does not store any of the data being tracked, and thus it is intended for uses such as experiencing new exercises, verifying whether the environmental conditions are consistent with the Kinect sensor requirements, visible space or adequate placement. When finished, the user is redirected to the main menu, with no data being stored.

6.3 Data Storage and Retrieval

One of the most significant components of the framework is the ability to store and retrieve data. Without this component, basic operations such as identifying users and exercises, defining keyframes or evaluating the user performance would not be possible. In some cases, a great amount of data may also need to be stored and retrieved. It is, therefore, critical to have a well structured way of dealing with it at any moment.

- User data - in order to identify the patient in the interface, it was relevant to store basic information such as the name and an identification number, included in a local database. Other stored information includes the user execution data which comprises, frame by frame, the data tracked during the execution of a certain exercise. The user collected data, resulting from the performance evaluation of the originally recorded exercise with the user execution of that same exercise, is also stored using an XML file.
- Exercise data - each recorded exercise required essentially three kinds of data: the identification data enabled to store basic exercise information, such as a generic name and identification number, data resulting from the recorded movements, and finally keyframes data, defined during the second step of the exercise creation process, and comprising information such as the angles threshold and joints weight.

Chapter 7

Framework Implementation

Following the framework architecture and design, detailed in Chapter 6, this chapter focus upon the implementation decisions of the specified framework architecture from a developer's perspective. Each of the components of the framework implementation are shown and described in greater detail, as well as the main challenges faced during the implementation stage.

7.1 Exercise Creation

In order to create an exercise, the first required step after selecting the option in the menu interface, was to detect whether there was a user in the visible space. These user detection followed by avatar joints mapping are performed by the ZigFu framework, based upon the NiTE algorithms. By the time the user was detected, though, a time frame before the framework started storing the tracked data was included. This was included so that the user could get acquainted with the three-dimensional space of the scene, avatar positioning and joint orientation. Listing 7.1 shows an example of this, where the user movements are always being tracked and executed, though not stored, until the *countdownTime* reaches 0. In listing 7.2, the same is done in the Update() function, with the addition of the methods that enable data to be stored in a local XML file. However, only active joints, i.e., joints that are actually being used to map the avatar movements.

When the recording task is finished, triggered by the clicking an on screen "Finish" button, follows the "Editing Exercise" step. In order to display the graphical interface of this component, the *UnityEngine.GUI* features were used, along with a *KeyframeClass* class, shown in listing 7.3 to keep the joints weight and threshold while these parameters are being edited in the graphical interface. Once the "ADD" button is clicked, the keyframe parameters are added to a *SortedList<int, KeyframeClass>*, where instances of KeyframeClass are kept along with the respective frame it is being associated. When the user clicks "FINISH", the sorted list of keyframes are stored in the XML file, using a method with a behaviour similar to other the one used in methods previously covered.

```
void Zig_UpdateUser(ZigTrackedUser user)
{
  if(countdownTime > 0 && play){
    countdownTime -= Time.deltaTime;

    UpdateRoot(user.Position);

    if(user.SkeletonTracked)
        {
            foreach (ZigInputJoint joint in user.Skeleton)
            {
        if (joint.GoodPosition)
          UpdatePosition(joint.Id, joint.Position);
                if (joint.GoodRotation)
          UpdateRotation(joint.Id, joint.Rotation);
            }
        }
  }

  // ...
}
```

Listing 7.1: Countdown until start recording

Figure 7.1: Countdown time until exercise starts

```
1  void Zig_UpdateUser(ZigTrackedUser user)
2  {
3    // ...
4
5    UpdateRoot(user.Position);
6    writeRoot(frameId, user.Position);
7
8    if (user.SkeletonTracked){
9      foreach (ZigInputJoint joint in user.Skeleton){
10       if(aj.isActiveJoint((int)joint.Id))
11         writeJoint((int)joint.Id, joint.Position, joint.Rotation);
12       if (joint.GoodPosition)
13         UpdatePosition(joint.Id, joint.Position);
14       if (joint.GoodRotation)
15         UpdateRotation(joint.Id, joint.Rotation);
16     }
17     writer.WriteEndElement();
18   }
19   frameId++;
20 }
21
22 public void writeJoint(int jointId, Vector3 position, Quaternion rotation){
23   writer.WriteStartElement("joint");
24   writer.WriteAttributeString("id", jointId.ToString());
25
26   writer.WriteStartElement("position");
27   writer.WriteAttributeString("x", position.x.ToString());
28   writer.WriteAttributeString("y", position.y.ToString());
29   writer.WriteAttributeString("z", position.z.ToString());
30   writer.WriteEndElement();
31
32   writer.WriteStartElement("rotation");
33   writer.WriteAttributeString("x", rotation.x.ToString());
34   writer.WriteAttributeString("y", rotation.y.ToString());
35   writer.WriteAttributeString("z", rotation.z.ToString());
36   writer.WriteAttributeString("w", rotation.w.ToString());
37   writer.WriteEndElement();
38
39   writer.WriteEndElement();
40 }
41
42 public void writeRoot(int frameId, Vector3 root){
43   writer.WriteStartElement("frame");
44   writer.WriteAttributeString("id", frameId.ToString());
45   writer.WriteAttributeString("x", root.x.ToString());
46   writer.WriteAttributeString("y", root.y.ToString());
47   writer.WriteAttributeString("z", root.z.ToString());
48 }
```

Listing 7.2: Storing recorded data

```
1  public class KeyframeClass{
2    public int[] jointsThreshold;
3    public int[] jointsWeight;
4
5    public KeyframeClass (int activeJointsCount){
6      jointsThreshold = new int[activeJointsCount];
7      jointsWeight = new int[activeJointsCount];
8    }
9  }
10
11 // ...
12
13 if(GUI.Button(new Rect(Screen.width/2, 2*(Screen.height/3), rw, rh), "ADD"))
14   keyframesList.Add(currentFrame, kc);
15
16 // ...
17
18 if(GUI.Button(new Rect(Screen.width/2+150, 2*(Screen.height/3), rw, rh), "FINISH"))
19   writeKeyframes(keyframesList);
```

Listing 7.3: Storing keyframe parameters

7.2 Exercise Execution

As seen in the previous chapter, the execution may be continuous or paused. In terms of implementation, this required a few different lines of code be executed during keyframes. For the continuous execution, no interruption in the avatar movement performing the original exercise had to be implemented. However, it requires a function to keep track of the best angle resulting from the comparison between the average rotations of the original exercise and the performed rotations for a given number of frames after the keyframe occurs, as shown in Listing 7.4. While the framesThreshold is not reached, the best obtained average angle until that moment is stored and updated if a better one is achieved. When the frames threshold limit is reached, the resulting angle is written to the XML file.

Paused execution behaves differently in that the end of a keyframe comparison is triggered not by a frames threshold, but by a correct pose performance, i.e., by having all resulting angles of weighted joints to be within the respective thresholds. Ultimately, the keyframe comparison ends in case the user is not able to perform the pose within a timeout, storing the best achieved average angle in a XML file with the user collected data, as can be seen in listing 7.5.

Both execution modes output a visual feedback of each joint's performance. To do this, a gradient color scheme was used from green to red, i.e., the RGB values ranged from (0,1,0), when the resulting angle was less than or equal to the threshold, to (1,0,0), in case the resulting angle was greater than the double of the threshold. Listing 7.6 shows how this was addressed.

```
if(framesThresholdCounter > framesThreshold){
  writeContinuousKeyframe(bestAngles, keyframesList[keyframeID].jointsWeight);

  lastKeyframeAvgAngle = bestAvgAngle;
  bestAvgAngle = int.MaxValue;
  framesThresholdCounter = 0;
  keyframeComparison = false;
}
else{
  float[] angles = GetAngles(keyframeJoints, userJoints);

  for(int i = 0; i < angles.Length; i++){
    if(keyframesList[keyframeID].jointsWeight[i] > 0){
      currentAvgAngle += angles[i] * keyframesList[keyframeID].jointsWeight[i];
      weightCounter += keyframesList[keyframeID].jointsWeight[i];
    }
  }
  if(weightCounter > 0)
    currentAvgAngle /= weightCounter;
  if(currentAvgAngle < bestAvgAngle){
    bestAngles = angles.Clone();
    bestAvgAngle = currentAvgAngle;
  }

  framesThreshold++;
}
```

Listing 7.4: Frames threshold in continuous execution

```
float[] angles = sc.GetAnglesFrame(keyframeJoints, userJoints);
timeCounter += Time.deltaTime;
play = true;

for(int i = 0; i < angles.Length; i++){
  if(keyframesList[currentFrame].jointsWeight[i] > 0){
    currentAvgAngle += angles[i] * keyframesList[currentFrame].jointsWeight[i];
    weightCounter += keyframesList[currentFrame].jointsWeight[i];
  }

  if(keyframesList[currentFrame].jointsWeight[i] > 0 && angles[i] > keyframesList[
      currentFrame].jointsThreshold[i])
    play = false;
}
if(weightCounter > 0)
  currentAvgAngle /= weightCounter;
if(currentAvgAngle < bestAvgAngle){
  bestAngles = angles.Clone();
  bestAvgAngle = currentAvgAngle;
}

if(play || timeCounter > timeLimit){
  if(play)
    bestAngles = angles.Clone();

  writePausedKeyframe(timeCounter, bestAngles, keyframesList[currentFrame].
      jointsWeight);

  timeCounter = 0.0f;
  bestAvgAngle = int.MaxValue;
  play = true;
}
```

Listing 7.5: Angles threshold in paused execution

```
if(angle <= threshold)
  colorStyle.normal.textColor = new Color(0,1,0);

if(angle > threshold){
  double r = 1 - ( (threshold * 1.5 - angle) / (threshold * 1.5 - threshold) );
  colorStyle.normal.textColor = new Color((float)r,1,0);
}

if(angle > threshold * 1.5){
  double g = (threshold * 2 - angle) / (threshold * 2 - threshold * 1.5);
  colorStyle.normal.textColor = new Color(1,(float)g,0);
}

if(angle > threshold * 2)
  colorStyle.normal.textColor = new Color(1,0,0);
```

Listing 7.6: Angles gradient color scheme

```
1  public DataTable GetExercises(){
2    return LoadData("SELECT * FROM Exercise");
3  }
4
5  public DataTable GetUsers(){
6    return LoadData("SELECT * FROM User");
7  }
8
9  public DataTable GetUserExecutions(int userId){
10   return LoadData("SELECT * FROM UserExercise WHERE userId = " + userId);
11 }
12
13 public void AddExercise(String exerName){
14   String query = "INSERT INTO Exercise(exercisename) VALUES('" + exerName + "')";
15   AddData(query);
16 }
```

Listing 7.7: C# methods used to access SQLite database

7.3 Data Storage and Retrieval

The different storage methods were used in this framework, both SQLite database and XML file, depending on the type of data. In order to store and retrieve data from/to the SQLite database, a C# external library was used, *Finisar.SQLite*. It is an ADO.NET data provider library for .NET languages and comprises a set of classes that enable tasks such as connecting to a local database file (.db) and sending C# strings as SQL statements. All basic SQL commands are supported, including "SELECT" or "UPDATE". Data extracted from the database is returned in a *System.Data.DataTable* class type, with the *DataRowCollection* including all entries, each one in a *DataRow*. Listing 7.7 comprises some of the most commonly used methods for database data storage and retrieval.

To store and retrieve data using XML files, a different approach had to be used. The *System.Xml* includes some relevant classes that enable to read from and write to a local XML file to be done easily, using *XmlReader* and *XmlWriter* class types.

7.3.1 User Data

The user data is stored in a local SQLite database, as seen in figure 7.2 and comprises:

#	Name	Data type	P	F	U	H	N	C
1	userId	INTEGER						
2	username	VARCHAR						

Figure 7.2: User database table

```
<exercise>
  <frame id="60" x="-138.0824" y="-12.71529" z="-2262.142">
    <joint id="1">
      <position x="-103.1414" y="723.0038" z="-2247.241" />
      <rotation x="-0.01980173" y="0.08803365" z="-0.006502321" w="0.9958994" />
    </joint>
    <joint id="2">
      <position x="-105.2187" y="503.7382" z="-2238.334" />
      <rotation x="-0.01980173" y="0.08803365" z="-0.006502321" w="0.9958994" />
    </joint>
  </frame>
</exercise>
```

Listing 7.8: Example of exercise data XML structure

- user name - the name that will be visible in the user selection screen of the framework. It is a VARCHAR field type;

- user identifier - a unique user identifier that will be necessary in order to associate exercise executions to the user. It is an INTEGER field type.

7.3.2 Exercise Data

The exercise data comprises the outcome of the first step of the exercise creation mode. All recorded data is stored in a single local XML, as shown in listing 7.8, file and comprises:

- frame element - number of the frame in the whole exercise, represented by the *id* included in the *frame* element; the frame identifiers are incremental over the course of the time, i.e., the *id* of the first frame is 0, the second frame is 1, etc. The three-dimensional space position of the user is stored in a set of three attributes, *x*, *y* and *z*, included in the *frame* element. It uses a referential based upon the sensor visible space to approximately refer to the center of gravity of the tracked human body.

- joint element - identifies the corresponding joint using a specific sequence of joints internally defined by the ZigFu API; the *id* attribute is included in each *joint* element, which is a child element of *frame*. Only joints actively tracked by the framework are stored in the XML file.

- joint position element - *position* is a child element of *joint* and includes the *x*, *y* and *z* attributes, referencing the three-dimensional space position of the corresponding joint during in that frame.

- joint rotation element - similarly to the *position* element, the *rotation* element is a child element of *joint* and includes the *x*, *y*, *z* and *w* attributes, corresponding to the quaternion values required to calculate the three-dimensional rotation of the joint.

```
<evaluation exerciseid="103" paused="False">
  <keyframe frameid="47">
    <joint jointid="1" threshold="0" weight="0" />
    <joint jointid="2" threshold="0" weight="0" />
    <joint jointid="3" threshold="0" weight="0" />
    <joint jointid="6" threshold="25" weight="2" />
    <joint jointid="7" threshold="25" weight="2" />
    <joint jointid="12" threshold="25" weight="2" />
    <joint jointid="13" threshold="25" weight="2" />
    <joint jointid="17" threshold="15" weight="1" />
    <joint jointid="18" threshold="15" weight="1" />
    <joint jointid="21" threshold="15" weight="1" />
    <joint jointid="22" threshold="15" weight="1" />
  </keyframe>
</evaluation>
```

Listing 7.9: Example of keyframe data XML structure

A class in the SQLite database, illustrated in figure 7.3, stores basic exercise data in order to enable exercises to be easily listed and accessed.

- exercise identifier - unique exercise identifier of the INTEGER field type.
- exercise name - the name of the exercise is useful to easily identify it when selecting an exercise to execute in the framework interface.

#	Name	Data type	P	F	U	H	N	C
1	exerciseId	INTEGER						
2	exercisename	VARCHAR						

Figure 7.3: Exercise database table

7.3.3 Exercise Keyframes Data

This is the data resulting from the second step of the creation mode, where the exercise keyframes are defined. All data is stored in a single local XML file, structured as shown in listing 7.9, with the filename being the corresponding exercise identifier.

- exercise element - *exercise* is the root element of the file and includes the exercise identifier *exerciseid* attribute and the boolean *paused* attribute, which defines whether the exercise is paused or continuous.
- keyframe element - child element of *exercise* that includes a *frameid* attribute referencing the frame identifier of the original exercise data. It is significant for the framework to be able to identify what data to perform the comparison.

- joint element - child element of *keyframe* containing the joint identifier *jointid* attribute, as well as the *threshold* and *weight* attributes. The former refers to the joint's angle threshold in degrees and the latter refers to the weight of the joint in the evaluation process. Each *keyframe* element includes a set o *joint* elements equal to the number of joints actively tracking the user movements.

7.3.4 User Execution Data

The user execution data XML adopts the same structure detailed in 7.3.2, though storing the data that resulted of the user attempt in performing the exercise. The SQLite database class of the execution, shown in figure 7.4 comprises the following fields:

- execution identifier - a unique identifier for each attempt of the user in performing the exercise and the primary key of the class; it was relevant to define this field as the primary key given that a user is able to have multiple executions of the same exercise on the same date.
- user identifier - identifier of the user responsible for this execution as an external key os the class.
- exercise identifier - identifier of the exercise the user attempted to perform as an exeternal key of the class.
- execution date - date of the execution, represented in the MM/DD/YYYY format.

#	Name	Data type	P	F	U	H	N	C
1	executionId	INTEGER						
2	userId	INTEGER						
3	exerciseId	INTEGER						
4	date	VARCHAR						

Figure 7.4: User execution database table

7.3.5 User Collected Data

The data collected about the user performance during an execution is stored in a single XML file, whose filename references the execution identifier. Listing 7.10 shows an example of the XML structure, which contains the following elements:

- frame element - contains the *id* attribute of the referencing the frame identifier of the original exercise data. A second *time* attribute stores the lapsed time in seconds the user required in order to perform the pose; this attribute is valid only for paused execution.
- joint element - child element of *frame* that features the *id* attribute, the joint identifier, and the *angle* attribute, which contains the angle performance in degrees for that joint. Only joints whose weight was greater than 0 are stored.

```
<exercise>
  <frame id="277" time="12.06958">
    <joint id="6" angle="24" />
    <joint id="7" angle="14" />
    <joint id="12" angle="24" />
    <joint id="13" angle="25" />
  </frame>
</exercise>
```

Listing 7.10: Example of user collected data XML structure in a paused exercise

Chapter 8

Validation and Results

The previous chapters have described the architecture, design and implementation of the framework in its final development stage. However, all tests performed during the iterative development process were not conducted with the intervention of end-users, i.e., children with hemiparesis or spastic diplegia, to provide any kind of feedback of the current state of the framework, mainly due to availability issues. It was only by the time the framework development was finished that a test group was gently provided by a clinic specialised in physical rehabilitation of children with motor impairments. Hence, this chapter details the test group, how the evaluation was conducted, the outcome and relevance of the collected results.

The validation process was carried out at the aforementioned physical rehabilitation clinic. The clinic provided a room featuring all the environmental requirements: it had enough unobstructed space for the children to perform the exercises, featured the appropriate lighting conditions and had the required infrastructures to properly place the Kinect sensor, as well as the video projector. Figure 8.1 illustrates the floor plan of the room where the tests took place.

The participants consisted of five children aged between 8 and 12 years old. According to the expanded and revised Gross Motor Function Classification System, all five participants are included in the Level I degree. Four of them had spastic diplegia, ranging from almost unnoticeable motor disorders to the naked eye (participants C and D), to a light severity degree disability (participants A and B), with visible disorder in the gait and flexion of the legs at the hips and knees, though perfectly able to walk without requiring further assistance. The hemiparesic child (participant E) featured a mild weakness of the limbs on the left side of the body. Among the participants, C and E were the highest, and D the lowest. Each session took place at a different time and were attended by the respective child's physiotherapist and legal guardian, which agreed with the goals of the tests conducted. On average, each session lasted no longer than 20 minutes, and only a single session was required for each participant.

The tests consisted of requesting the participants to perform each of the four exercises that were previously addressed and detailed, in the following order: raising the arms above the head, side steps, sequence of poses and scissors jump. As the participants had no previous contact with the framework, they were asked to perform each of the exercises one first time in order to get

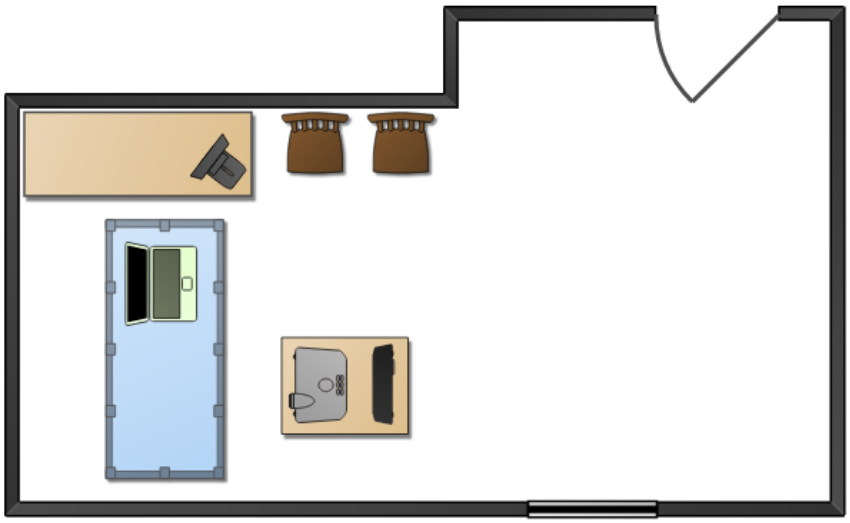

Figure 8.1: Test room floor plan

acquainted with the Kinect sensor, suggested exercises and the framework's virtual environment. For these tests, only the "Load Exercise" play mode was used, which enabled the child to embody an avatar with its movements. The goal was to mimic another character featured on screen, which was executing the originally recorded exercise, according to their physical capabilities. Along with the instructions given throughout this stage, all children almost immediately guessed what to do and how to achieve it. The results of this preliminary stage were then discarded and no video was recorded.

Once the preliminary stage was finished, the main test took place with the exact same setting and agenda as the previous step, with the only difference being the video recording and use of the collected results.

8.1 Raising the arms above the head

The first exercise to be executed was "Raising the arms above the head", due to its greater simplicity. Kinect's tracking accuracy was mostly positive in this slower exercise, which did not require complex joint orientation calculus. According to the results gathered in figure 8.2 most participants had a worth noting performance, with no relevant disparities in the angles results, though participants A and B showed a greater gap to the threshold angles. Participant E's right arm had a better performance result, which is consistent with his left-sided hemiparesis, while participant C had the best performance overall.

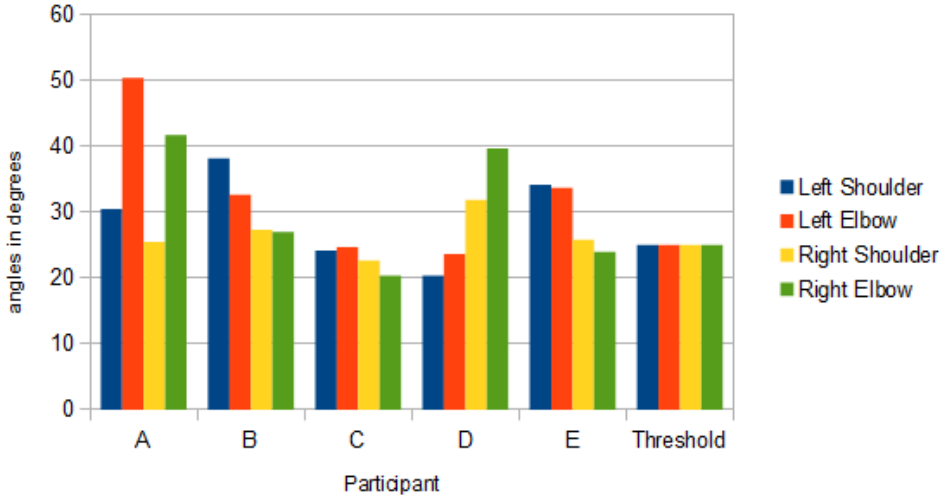

Figure 8.2: "Raising the arms above the head" exercise - average angles results on keyframes

8.2 Side steps

Followed the "side steps" exercise, whose collected results are presented in figure 8.3. While this is an exercise that did not impose great tracking complexity for the Kinect device, some participants became confused with the number of steps to take; this obstacle combined with the uninterrupted execution and considerable exercise speed, led to undesired and misleading results, most prominent in the data collected regarding participants A and D, which weren't able to completely follow the side steps. Nevertheless, participant C had one of the most positive results, along with participant E; once again, the right-sided limb of the latter evidences a better result than the left-sided one.

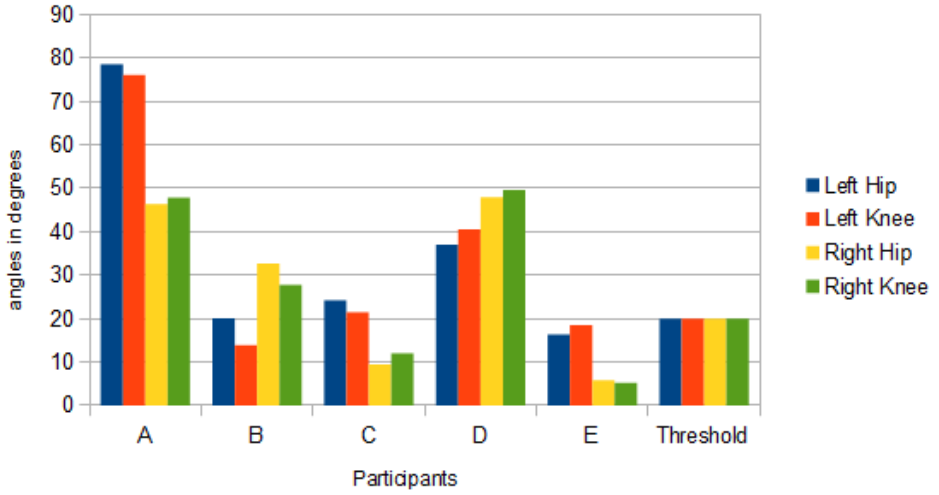

Figure 8.3: "Side steps" exercise - average angles results on keyframes

8.3 Sequence of poses

The third exercise consisted of "sequence of poses", the only paused exercise of the set. The participants were given a time limit of 30 seconds to achieve each of the poses, though this time around the required movements were of greater complexity, with certain poses even unachievable by some of the participants. However, along with greater execution complexity came greater tracking inaccuracy, mostly due to incorrect detection of the arms orientation. In some cases, this affected the participants performance results, since there were times where some joints would output a negative angle result (i.e., angle values greater than the thresholds), as opposed to the observed correct execution of the participant. Despite this, the feedback perceived by the individuals assisting the session seemed to be more encouraging during this specific exercise, which can probably be justified due to the continuous presence of an evaluation of the angles performance, delivered by an on screen gradient color scheme for each evaluated joint, ranging from green (good) to red (bad). Regarding the results displayed in figure 8.4, participant E's right arm performed slightly better than the left arm and, along with participants B and C, delivered a relevant outcome when compared with the established threshold. While participant B had the best average time, all of the participants were able to fully perform at least one of the four poses within the imposed time limit.

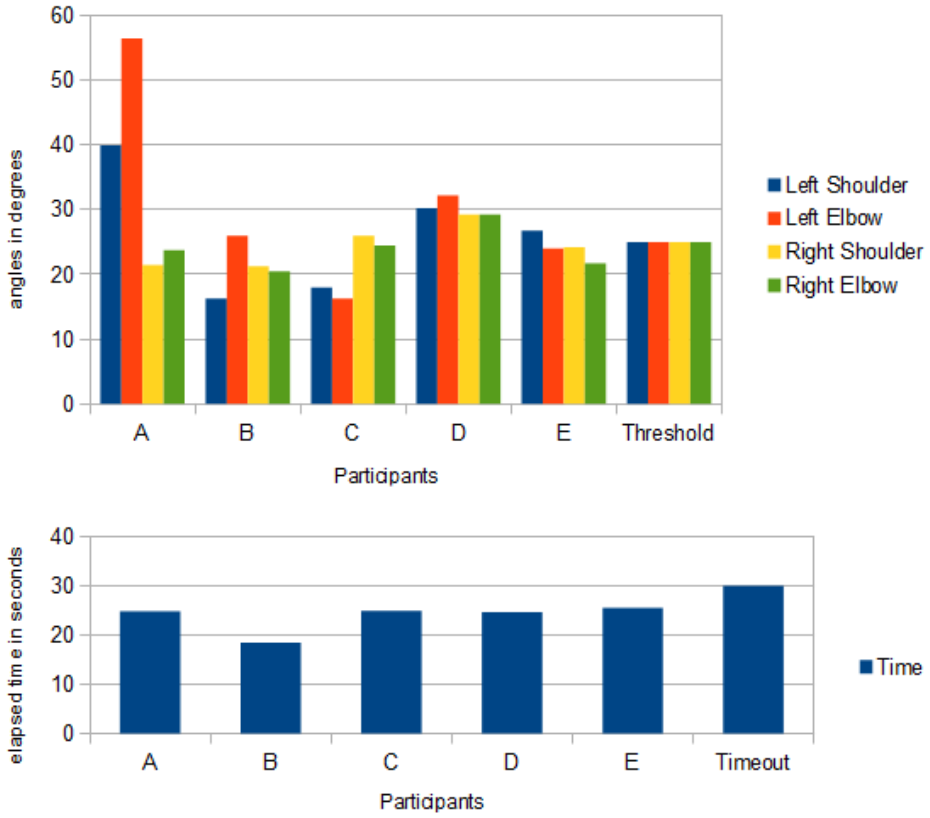

Figure 8.4: "Poses sequence" exercise - average elapsed time and angles results on keyframes

8.4 Scissors jump

The "scissors jump" exercise was held at the session's end. Feedback perceived by the involved participants led to the conclusion that this was arguably the most complex exercise, not only due to its fast paced nature, but also because it required considerable limbs coordination and increased abduction and adduction angles for both the arms and legs. This led to a more evident delay in effectively mimicking the avatar movements, as well as a greater difficulty in performing each of the keyframed movements as intended. Hence, the collected results shown in figure 8.5 are substantially worse than the ones observed in previous exercises. As expected though, for all participants, the joints associated with the legs had lower angle values, mainly due to a smaller abduction/adduction movement in degrees of the lower limbs in respect to the upper limbs. As would be expected, the evaluation results once again support that participant E's right-sided limbs performed better.

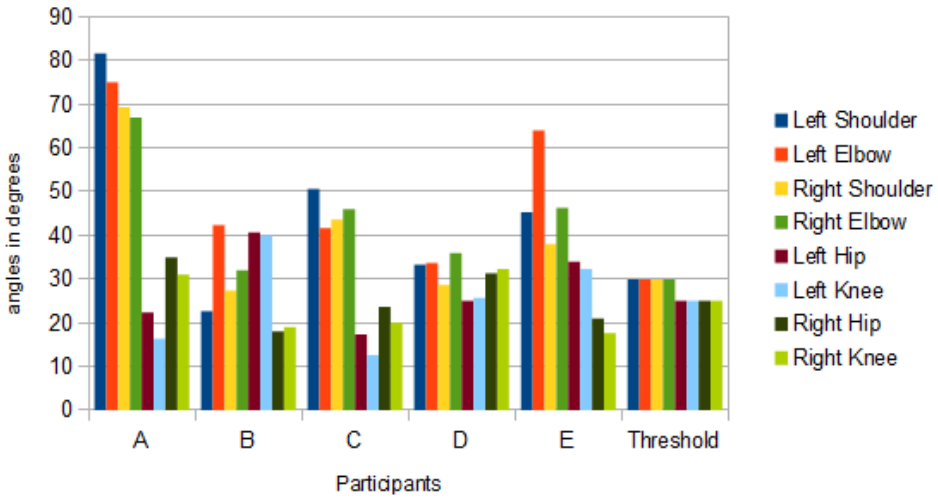

Figure 8.5: "Scissors jump" exercise - average angles results on keyframes

8.5 Discussion

The obtained results evidence some interesting patterns, some of them closely related with the motor limitations of each of the participants and the effort put in performing the exercises. Figure 8.6 shows the results obtained by aggregating all exercises together and calculating the average sum of the angles for every joint of each participant. On behalf of the participants characteristics, this chart evidences that Participant E had better results for both right-sided limbs; participant C obtained the overall best performance; and participant A had the overall less positive results. The participants B and D situations are a little bit more particular: while D's results are still notable, it is believed that his reduced height may have had some impact in detecting accurately at all times;

despite his condition, B was one of the most energetic and perspicacious participants, which may have influenced the outcome positively. However, a few issues observed are worth pointing out:

- Some detection inaccuracies noticed mainly during the sequence of poses exercise, due to the complex nature of the proposed joints orientation.
- Lack of joints performance visual feedback. Since the angles results were all being listed and displayed in the right side of the screen, sometimes participants seemed not well aware which joints were performing better or worse, only having a generic idea whether they were doing things correctly or not by retaining the overall colors balance.
- Some difficulties in keeping track of the exercise pace during continuous execution. This was most notably observed during the scissors jump exercise, but also occurred in side steps.

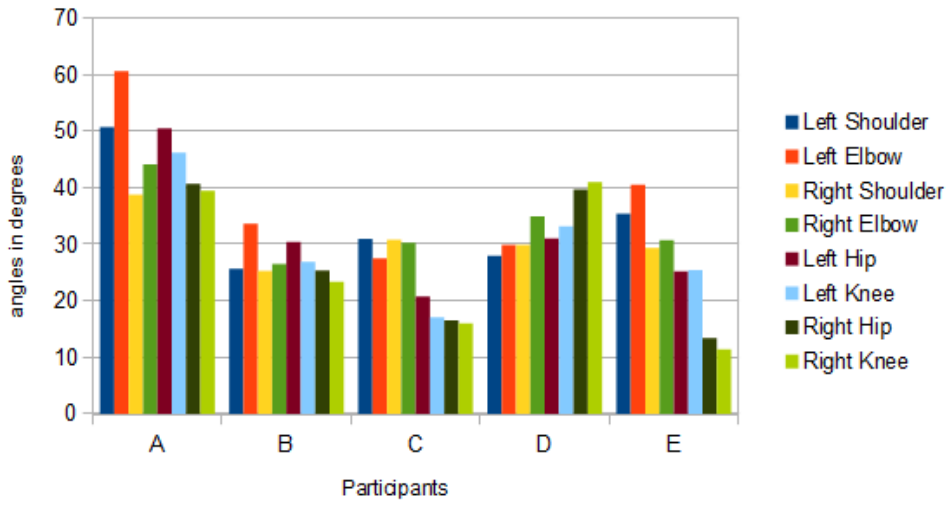

Figure 8.6: Aggregated angle results

Chapter 9

Conclusions and Future Work

The research and work delivered in this thesis led to the development of a framework for Kinect sensor to assist children with spastic diplegia and hemiparesis in their rehabilitation process. The tools provided with the framework enabled it to be explored in other contexts with different exercises and target-audience, due to its significant flexibility in providing evaluation metrics. The use of the Unity editor along with the ZigFu framework provided a significant range of opportunities in delivering a framework that would be consistent with the proposed goals. While the Kinect sensor certainly has potential to be used in the motor rehabilitation context and even further explored, some of its limitations can not be overlooked, mainly the tracking feasibility issues. These limitations narrow the potential of it being a device fit to be more widely applied in the motor rehabilitation context, though the results obtained evidence that it is possible to take advantage of its unique tracking capabilities to deliver a tool that, at the very least, impaired children with disorders equivalent to those of the tests participants are able to benefit from.

As a future work perspective, there are some aspects in which the framework could be improved, though had to be left out mainly due to time restrictions. Follows some of the changes and new additions that emerged during the development and evaluation stages, that not only emerged from the team that closely collaborated in this study, but that was also perceived by observing the participants behaviour to the framework, as well as physiotherapists remarks:

- A clearer way to display the angles results during paused execution's keyframes. One of the most obvious issues in providing an adequate visual feedback was that at times the participants weren't well aware as to which of the body joints each of the angles results matched. Although the color scheme helped them understanding whether they were doing it right or not, it would have been more helpful if each of these was a coloured circle attached to the respective joint of the participant's avatar.

- Voice instructions throughout the exercise, assisting the patient in understanding what movements to perform and pointing corrections to the performance during keyframes. This was mainly due to some perceived difficulties in understanding the actual avatar movements in the three-dimensional scene.

- A renewed menu navigation structure with support for motion controls, in order to make it consistent with the remaining modules of the framework and improve in terms of usability.
- Test the framework with an increased number of exercises and eventually broaden the spectrum of comprised motor disorders.
- A framework component capable of delivering a comprehensive way of representing and interpreting the collected user data using charts to show progress overtime.

Regarding the scenario of changing the used technologies, Microsot has announced a new generation of Kinect Sensor devices, including a Windows version scheduled for a 2014 release. Although no technical specifications were provided, according to official press releases and live demonstrations the sensor will be capable of providing higher accuracy and precision, an expanded field of view, improved skeletal tracking with an increased number of joints and a new IR sensor that expands the supported lighting conditions [Hed13]. This addresses nearly all the previously regarded Kinect sensor limitations, thus anticipating a bright future in the body motion tracking field.

References

[3dc] 3dcgi. 3d cameras. http://www.3dcgi.com/cooltech/cameras/cameras.htm.

[Ard] Arduino World. Gyro & accelerometers. http://www.fact4ward.com/blog/segway-clone/wii-motion-plus/.

[Bar10] Colin Barras. Microsoft's body-sensing, button-busting controller. http://www.newscientist.com/article/mg20527426.800-microsofts-bodysensing-buttonbusting-controller.html, 2010.

[Ber12] Paula Bernier. The call for speech recognition grows louder. http://www.tmcnet.com/voip/departments/articles/284125-call-speech-recognition-grows-louder.htm, 2012.

[Bla12] Joshua Blake. *Natural User Interfaces in .NET*, volume Part 1: Introducing NUI concepts - 1 The natural user interface revolution. Manning Publications Co., 2012.

[Bog07] Ian Bogost. The prehistory of wii fit,. http://www.bogost.com/watercoolergames/archives/the_prehistory.shtml, 2007.

[Car] Carnegie Mellon University ETC Wiki. Sony playstation move - unity 3d. http://wiki.etc.cmu.edu/unity3d/index.php/Sony_PlayStation_Move.

[Car08] Frank Caron. Of gyroscopes and gaming: the tech behind the wii motionplus. http://arstechnica.com/gaming/2008/08/wii-motion-sensor/, 2008.

[CCH11] Yao-Jen Chang, Shu-Fang Chen, and Jun-Da Huang. Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. *Research in Developmental Disabilities*, 32(6):2566–2570, 2011.

[Cha] Channel 9. Managed library for nintendo's wiimote. http://channel9.msdn.com/coding4fun/articles/Managed-Library-for-Nintendos-Wiimote.

[Cha12] Matthew Chapman. Motion-controlled games double market share. http://www.marketingmagazine.co.uk/news/1110750/Motion-controlled-games-double-market-share/, 2012.

[Cod] CodePlex. Managed library for nintendo's wiimote. http://wiimotelib.codeplex.com/.

[Con] Contemplas GmbH. Vicon motus 2d calculations. http://www.motus10.com/motion_analysis_motus_2d.aspx.

REFERENCES

[CoV] CoVii. Covii - viim. http://www.covii.pt/viim/.

[CWi] CWiid. Cwiid library. http://abstrakraft.org/cwiid/.

[Dal09] Amanda J. Daley. Can exergaming contribute to improving physical activity levels and health outcomes in children? *PEDIATRICS*, 124(2):763–771, 2009.

[DDR] DDR Freak. About ddr. http://www.ddrfreak.com/aboutddr.php.

[dSMPL+12] Felipe Augusto dos Santos Mendes, Jose Eduardo Pompeu, Alexandra Modenesi Lobo, Keyte Guedes da Silva, Tatiana de Paula Oliveira, Andrea Peterson Zomignani, and Maria Elisa Pimentel Piemonte. Motor learning, retention and transfer after virtual-reality-based training in parkinson‘s disease - effect of motor and cognitive demands of games: a longitudinal, controlled clinical studys. *Physiotherapy*, 98:217–223, 2012.

[Ele] Electronic Arts. Ea sports active. http://www.ea.com/ea-sports-active.

[FM12] David Fiedler and Heinrich Müller. Impact of thermal and environmental conditions on the kinect sensor. *International Workshop on Depth Image Analysis at the 21st International Conference on Pattern Recognition (accepted for publication)*, 2012.

[FOR] FORTH. Kinect 3d hand tracking. http://cvrlcode.ics.forth.gr/handtracking/.

[FPT12] Rita Francese, Ignazio Passero, and Genoveffa Tortora. Wiimote and kinect: gestural user interfaces add a natural third dimension to hci. *AVI ’12 Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 116–123, 2012.

[Gia] Giant Bomb. Eyetoy: Kinetic screenshots, images and pictures. http://www.giantbomb.com/eyetoy-kinetic/3030-6739/.

[gl.] gl.tter. Wiiyourself! http://wiiyourself.gl.tter.org/.

[Glo] GlobalSensing Technologies. Gst api. http://www.openni.org/files/gst-api/.

[Han12] Hansard et al. *Time-of-flight cameras: Principles, Methods and Applications*. Springer, 2012.

[Har] John Hartford. Spastic diplegia - overview and considerations for children. http://cpfamilynetwork.org/blogs/cerebral-palsy-news-spastic-diplegia-overview-and-considerations-for-children.

[HBC+96] Thomas T. Hewett, Ronald Baecker, Stuart Card, Tom Carey, Jean Gasen, Marilyn Mantei, Gary Perlman, Gary Strong, and William Verplank. *Curricula for Human-Computer Interaction*. ACM SIGCHI, 1996.

[Hed13] Bob Heddle. The new generation kinect for windows sensor is coming next year. http://blogs.msdn.com/b/kinectforwindows/archive/2013/05/23/the-new-generation-kinect-for-windows-sensor-is-coming-next-year.aspx, May 2013.

[Hof05] Bryan Hoff. Ditch your mouse: Why you should be using a tablet. http://www.peachpit.com/articles/article.aspx?p=383855, April 2005.

REFERENCES

[Inv08] InvenSense. Invensense idg-600 motion sensing solution showcased in nintendo's new wii motionplus accessory. http://invensense.com/mems/gyro/documents/articles/071508.html, 2008.

[Kina] Kinect for Windows. Kinect for windows sensor components and specifications. http://msdn.microsoft.com/en-us/library/jj131033.aspx.

[Kinb] Kinect for Windows. Kinect sensor setup, requirements, support. http://www.microsoft.com/en-us/kinectforwindows/purchase/sensor_setup.aspx.

[Kinc] Kinect Support , Xbox.com. More about kinect sensor placement. https://support.xbox.com/en-US/xbox-360/kinect/sensor-placement.

[Kin12] Kinect for Windows Team. Near mode: What it is (and isn't). http://blogs.msdn.com/b/kinectforwindows/archive/2012/01/20/near-mode-what-it-is-and-isn-t.aspx, January 2012.

[Koh06] Chris Kohler. Out of control: The coolest and craziest videogame controllers of all time. http://www.1up.com/features/out-of-control?pager.offset=3, 2006.

[Koh09] Chris Kohler. The 15 most influential games of the decade. http://www.wired.com/gamelife/2009/12/the-15-most-influential-games-of-the-decade/all/1, 2009.

[Kum09] Mathew Kumar. Develop 2009: Scee's hirani reveals ps eye facial recognition, motion controller details. http://www.gamasutra.com/php-bin/news_index.php?story=24456, 2009.

[Lam12] Wayne Lam. Smartphones see accelerated rise to dominance. http://www.isuppli.com/Mobile-and-Wireless-Communications/News/Pages/Smartphones-See-Accelerated-Rise-to-Dominance.aspx, 2012.

[McC11] John McCutchan. Aspiring developers take note: Move.me unveiled at gdc. http://blog.us.playstation.com/2011/03/02/aspiring-developers-take-note-move-me-unveiled-at-gdc/, March 2011.

[Mic12] Microsoft Robotics. Kinect sensor. http://msdn.microsoft.com/en-us/library/hh438998.aspx, 2012.

[Mik09] Anton Mikhailov. Playstation motion controller interview part 2. http://www.viddler.com/v/8e5c8702, 2009.

[mJH12] Hui mei Justina Hsu. The potential of kinect as interactive educational technology. *ICEMT2011*, 13:334–338, 2012.

[Mos11] Sebastian Moss. Move.me: Developer talks about building drivers for the playstation move. http://www.playstationlifestyle.net/2011/05/12/move-me-developer-talks-about-building-drivers-for-the-playstation-move/, 2011.

[Nat12] National Institute of Neurological Disorders and Stroke. Cerebral palsy: Hope through research. http://www.ninds.nih.gov/disorders/cerebral_palsy/detail_cerebral_palsy.htm, August 2012.

REFERENCES

[Nina] Nintendo. Wii fit - game info. http://www.nintendo.com/games/detail/hoiNtus4JvIcPtP8LQPyud4Kyy393oep.

[Ninb] Nintendo. Wii sports - features. http://wiisports.nintendo.com/features_section/.

[Ninc] Nintendo. Wii sports - game info. http://www.nintendo.com/games/detail/1OTtO06SP7M52gi5m8pD6CnahbW8CzxE.

[Nin06] Nintendo. Wii sports instruction booklet (pal), 2006.

[Nin08a] Nintendo. Wii balance board operations manual, 2008.

[Nin08b] Nintendo. Wii fit instruction booklet (pal), 2008.

[Opea] Open Kinect Wiki. Protocol documentation. http://openkinect.org/wiki/Protocol_Documentation.

[Opeb] OpenNI. About openni. http://www.openni.org/about/.

[Opec] OpenNI. Openni sdk. http://www.openni.org/openni-sdk/.

[Per10] Sarah Perez. Weekend project: Hack microsoft kinect. http://readwrite.com/2010/11/19/weekend_project_hack_microsoft_kinect, November 2010.

[Pin11] Melanie Pinola. Speech recognition through the decades: How we ended up with siri. http://www.pcworld.com/article/243060/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html, 2011.

[Plaa] PlayStation. Eyetoy: Kinetic. http://uk.playstation.com/ps2/games/detail/item42723/EyeToy-Kinetic/.

[Plab] PlayStation. Move.me - software tool for ps3 that uses playstation move technology. https://us.playstation.com/ps3/playstation-move/move-me/.

[Pla11] PlayStation Blog. Move.me available today on playstation store, free for students and educators. http://blog.us.playstation.com/2011/07/26/move-me-available-today-on-playstation-store-free-for-students-and-educators/, 2011.

[PRBL07] Robert Palisano, Peter Rosenbaum, Doreen Bartlett, and Michael Livingston. Gross motor function classification system - expanded and revised. http://motorgrowth.canchild.ca/en/GMFCS/resources/GMFCS-ER.pdf, 2007.

[Pri] PrimeSense. Nite middleware. http://www.primesense.com/solutions/nite-middleware/.

[Qua] Quagmire's Kingdom. Review: Ea sports: Active (wii). http://www.kingquagmire.com/3014/review-ea-sports-active-wii/.

[Rau08] Megan Rauscher. Wii fit finding its way into rehab. http://www.reuters.com/article/2008/06/12/us-wii-fit-rehab-idUSTON27758620080612, 2008.

[RG01] J. Rodda and H. K. Graham. Classification of gait patterns in spastic hemiplegia and spastic diplegia: a basis for a management algorithm. *European Journal of Neurology*, 8:98–108, 2001.

REFERENCES

[Ric99] James G. Richards. The measurement of human motion: A comparison of commercially available systems. *Human Movement Science*, 18(5):589–602, October 1999.

[Rop05] Chris Roper. Eyetoy: Kinetic. http://www.ign.com/articles/2005/11/23/eyetoy-kinetic, 2005.

[SB06] Leonid Sigal and Michael J. Black. Humaneva: Synchronized video and motion capture dataset for evaluation of articulated human motion. 2006.

[SFC+11] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. *CVPR11*, 2011.

[Sig] SigmaRD. Sigmanil. http://www.sigmanil.com/.

[Sli12] Marty Sliva. The essential 100, no. 10: Wii sports. http://www.1up.com/features/essential-10-wii-sports, 2012.

[Sni08] Mike Snider. Wii finds home in retirement communities, medical centers. http://usatoday30.usatoday.com/tech/gaming/2008-05-14-wii-retirement-medical-centers_N.htm, 2008.

[SS11] Erik E. Stone and Marjorie Skubic. Evaluation of an inexpensive depth camera for passive in-home gait assessment. *Journal of Ambient Intelligence and Smart Environments*, 3(4):349–361, 2011.

[Tho08] Tor Thorsen. Q&a: Ea sports active-ating wii. http://www.gamespot.com/news/qanda-ea-sports-active-ating-wii-6201028, 2008.

[TL] Gabriel Taubin and Douglas Lanman. Structured light for 3d scanning. http://mesh.brown.edu/3dpgp-2009/homework/hw2/hw2.html.

[Tob] Tobii. Tobii pceye - eye control on your pc with your eyes. http://www.tobii.com/pceye.

[Tod09] Bred Todd. Ea sports active review. http://www.gamespot.com/ea-sports-active/reviews/ea-sports-active-review-6210438/, 2009.

[Unia] Unity. Unity - multiplatform. http://unity3d.com/unity/multiplatform/.

[Unib] Unity. Unity workflow. http://unity3d.com/unity/workflow/.

[Unic] Unity. What's new in unity 4.0. http://unity3d.com/unity/whats-new/unity-4.0.

[Wei10] Thomas C. Weiss. Hemiparesis - facts and information. http://www.disabled-world.com/health/neurology/hemiparesis.php, September 2010.

[Ziga] ZigFu. Zdk overview. http://zigfu.com/en/zdk/overview/.

[Zigb] ZigFu. Zigfu browser plugin. http://zigfu.com/en/downloads/browserplugin/.

[Zig12] ZigFu. Zdk for javascript documentation. http://zigfu.com/static/apidoc/files/zig-js.html, 2012.